MASTER EN SCIENCES INFORMATIQUES

Signatures digitales

Huart, Carl

Award date:
1986

Awarding institution:
Universite de Namur

Facultés Notre-Dame de la Paix , Namur

Institut d'informatique

Année académique 1985 - 1986

Signatures Digitales

Carl Huart

Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique

Je tiens à remercier,
Messieurs Jean Raemakers
Jean Jacques Quisquater
Yves Moulart
ainsi que le personnel de Bancontact qui m'ont, par leur précieuse collaboration, aidé à la rédaction de ce travail.

Chapitre 1 Introduction

Le problème posé par l'implémentation de signatures digitales, en vue de remplacer les systèmes de communications d'affaires actuels par des systèmes de téléprocessing, ne peut être résolu que par des techniques cryptographiques.
La cryptographie est l'étude des solutions mathématiques permettant de résoudre deux sortes de problèmes de sécurité : celui de la préservation de la confidentialité des communications et celui de l'authentification des messages échangés et/ou du correspondant.
La préservation de la confidentialité implique la mise en place d'un système empêchant des individus non autorisés de comprendre l'information échangée entre deux correspondants sur un moyen de transmission public.
L'authentification des messages empêche l'injection non autorisée d'information sur un moyen de transmission public, assurant ainsi le destinataire de la bonne provenance du message. La propriété de pouvoir certifier un message et identifier son expéditeur pourrait être utilisée comme substitut digital aux signatures manuscrites.
Pour implémenter un système d'authentification, il est nécessaire de se baser sur des algorithmes de cryptographie, qu'ils soient à clé publique ou à clé secrète.

Les chapitres 2 et 3 introduiront les algorithmes de cryptographie utilisés pour implémenter des signatures digitales.
Le chapitre 4 traitera des signatures proprement dites, en donnant la définition des différents types de signatures et les diverses méthodes de réalisation. L'importance que prennent les algorithmes de cryptographie dans ces méthodes, conduit à rechercher pour ceux-ci des améliorations notables en temps d'éxécution.
Les chapitres 5 et 6 définiront des optimisations à effectuer sur les algorithmes de cryptographie pour obtenir des implémentations efficaces de signatures digitales.
Le chapitre 7 étudiera un problème de sécurité relatif aux signatures digitales : celui des systèmes électroniques de transfert de fonds et plus particulièrement de **Bancontact**.

Le chapitre 8 décrira les méthodes d'implémentation choisies et les tables de tests indiquant les résultats obtenus.

Chapitre 2 Algorithme à clé secrète

2.1 Introduction

Les systèmes à clé secrète sont historiquement les plus anciens puisqu'on en retrouve trace dans l'Antiquité.
L'expéditeur possède un texte en clair **M**, qu'il désire faire parvenir de manière confidentielle au destinataire. Pour empêcher tout adversaire de connaître le contenu de **M**, l'expéditeur opère sur **M** une transformation inversible E_k, ayant pour but de produire un texte chiffré ou cryptogramme $C = E_k(M)$. Le paramètre **k**, appelé la clé, est transmis au destinataire via une ligne de communication sûre (cette ligne ne peut être utilisée pour transmettre **M** pour des raisons de capacité, de rapidité ou de coût). Comme le destinataire connait **k**, il peut déchiffrer **C** en calculant $D_k(C)$ pour obtenir ainsi le texte original du message. L'adversaire ne connaissant pas **k**, ne sera pas capable de comprendre **C**.

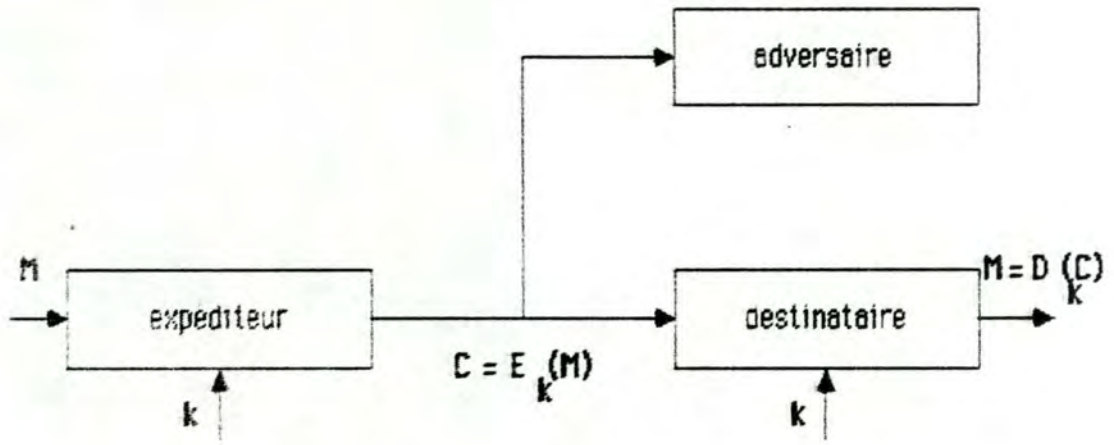


Figure 2.1 Schéma général d'échange de messages par système à clé secrète

Un système cryptographique à clé secrète est donc un ensemble $\{E_k\}$ de

transformations inversibles E_k : {M} -> {C} d'un ensemble {M} de messages en clair vers un ensemble {C} de messages chiffrés. La clé **k** est choisie dans un ensemble fini {**k**} de clés.
Le but à atteindre lorsqu'on crée un cryptosystème E_k est de rendre les opérations de chiffrement et de déchiffrement peu coûteuses tout en s'assurant que toute tentative de l'adversaire pour comprendre le texte chiffré serait trop complexe (en temps ou en nombre d'instructions requises) ou trop chère pour être praticable.

2.2 Data Encryption Standard

2.2.1 Introduction

Reconnaissant le besoin urgent de protection des données au sein du gouvernement américain et des entreprises privées, et percevant le fait que seuls le chiffrement et l'authentification étaient des moyens viables pour protéger les communications et les banques de données, le **National Bureau of Standards (NBS)** encouragea la création de données informatiques.
Pour le **NBS**, un algorithme acceptable devait au moins présenter les qualités suivantes :

- être complètement spécifié et non ambigü.
- fournir un niveau de protection calculable qui pourrait être exprimé en temps nécessaire pour découvrir la clé en ne connaissant que le texte chiffré ou que des morceaux de texte en clair et le texte chiffré.
- avoir une méthode de protection basée uniquement sur le secret de la clé.
- ne pas varier suivant les utilisateurs.

C'est ainsi que le 6 août 1974, **IBM** présenta un algorithme remplissant ces diverses conditions. Cet algorithme allait devenir la base du futur **DES**.

Le **DES** est également un standard **ANSI**, appelé **DEA** (Data Encryption Algorithm) et est utilisé dans des applications industrielles. Depuis peu, il est devenu un standard **ISO** sous le nom de **DEA1**.

2.2.2 Description

1) Schéma général de fonctionnement

Le DES reçoit en entrée 64 bits (le message en clair sera donc découpé en blocs de 64 bits) et fournit en sortie des blocs de 64 bits (pas d'expansion du contenu du message).
Dans la phase de permutation initiale, on réordonne simplement les bits en entrée selon une table de permutation : le premier bit devient le $40^{ème}$, le second devient le $8^{ème}$, ... (voir annexe 1.1).
On effectue ensuite la phase de chiffrement proprement dite.
On termine finalement par une permutation qui sera l'inverse de la permutation initiale (voir annexe 1.2).

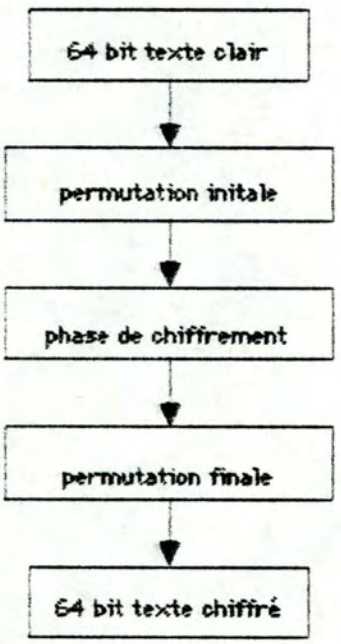


Figure 2.2 Diagramme général

2) Schémas détaillés

a) Phase de chiffrement

On divise les 64 bits en deux groupes de 32 bits appelés R_0 et L_0. K_1 sera une sous-clé générée directement à partir de la clé de chiffrement. On utilise une fonction f dont les paramètres d'entrée sont R_0 et K_1. Le résultat de cette fonction sera ajouté modulo 2 à L_0. On permute ensuite les registres R_0 et L_0. $L_0 \oplus f(R_0,K_1)$ devient R_1 et R_0 devient L_1. Cela termine une itération. On répète 16 fois l'opération consistant à additionner modulo 2 le contenu du registre gauche avec $f(R_i,K_{i+1})$ et swapper les registres gauches et droits de manière à obtenir une indépendance complète entre les 64 bits de texte clair et les 64 bits de texte chiffré.

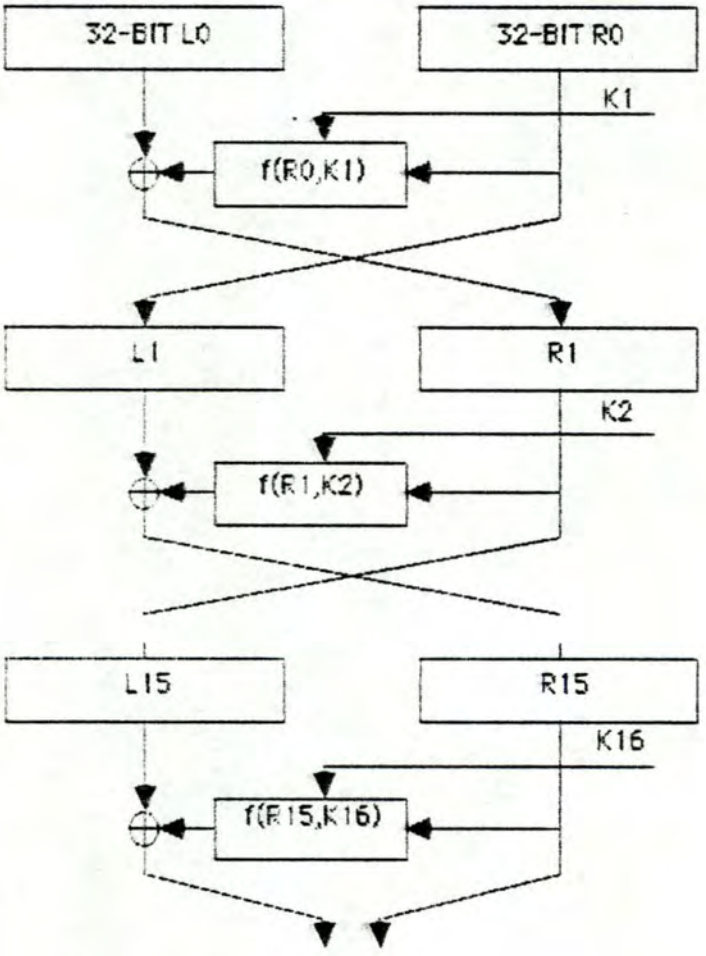


Figure 2.3 Diagramme de chiffrement

b) Génération des sous-clés K_1, K_2, K_3, ..., K_{16} à partir de la clé principale

On ne considère que 56 bits sur les 64 que compte la clé principale, les 8 autres bits seront utilisés comme bits de parité. La première transformation appliquée sur ces 56 bits s'appelle "**permuted choice1**". Il s'agit simplement de permuter les bits selon une table (voir annexe 1.3) et de les grouper ensuite en deux blocs de 28 bits appelés C_0 et D_0.

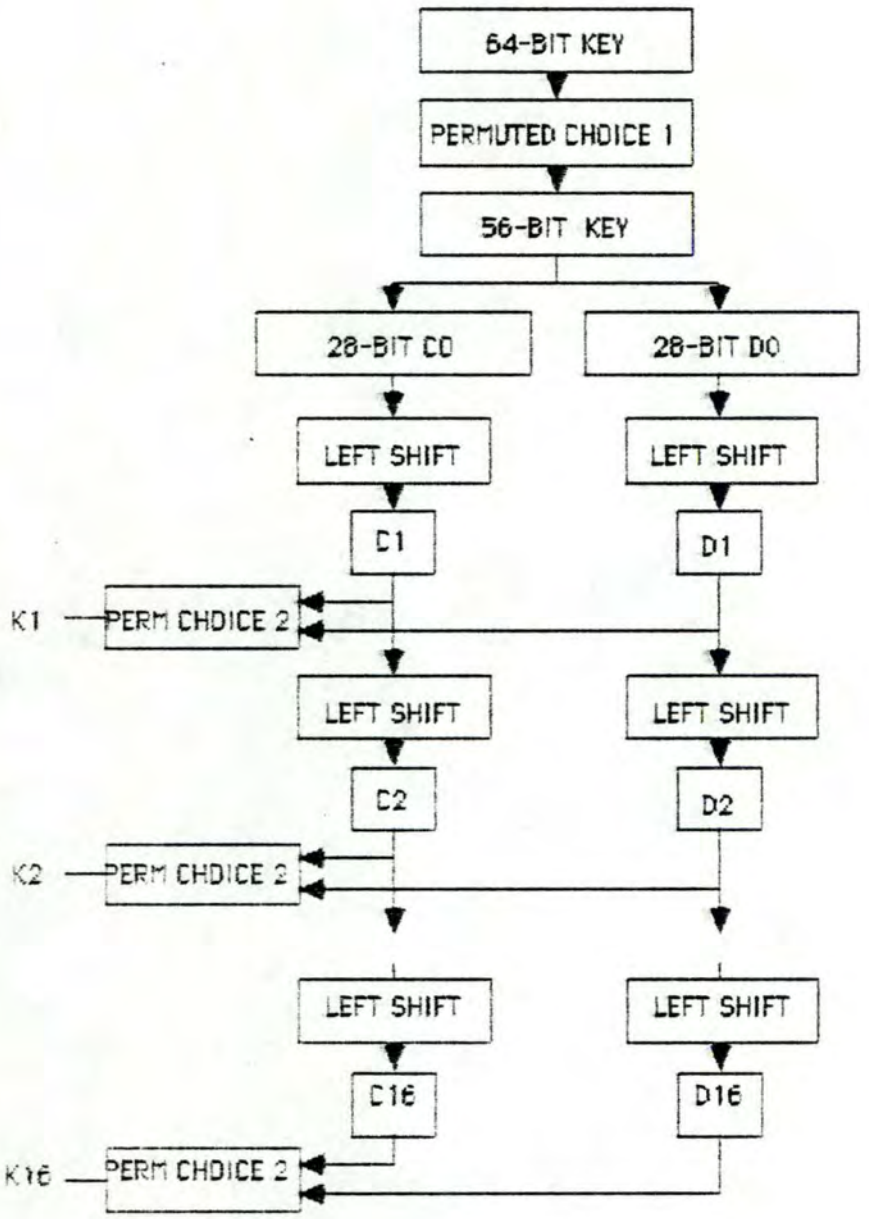


Figure 2.4 Génération des sous-clés à partir de la clé principale

On effectue sur C_0 et D_0 un shift circulaire gauche de un bit et on obtient ainsi C_1 et D_1. On permute les bits de C_1 et D_1 selon la table de "**permuted choice 2**" (voir annexe 1.4) pour générer la sous-clé K_1. Chaque sous-clé K_i sera obtenue de la même manière, sauf qu'on fera parfois un shift circulaire gauche de deux bits au lieu d'un bit (voir annexe 3).

c) Génération de la fonction **f**

La première opération réalisée est la permutation **E**. Il s'agit d'une permutation recevant 32 bits provenant du registre R_0 en entrée et fournissant 48 bits en sortie (voir annexe 1.5).
Les 48 bits résultant de la permutation sont ajoutés modulo 2 aux 48 bits de la sous-clé K_1. Le résultat est réduit à 32 bits par l'utilisation des fonctions de mapping $S_1, S_2, ..., S_8$ acceptant 6 bits en entrée et fournissant 4 bits en sortie (voir annexe 2)
Finalement la permutation **P** (voir annexe 1.6) est effectuée sur ces 32 bits. Le résultat de cette permutation donnera $f(R_0,K_1)$.

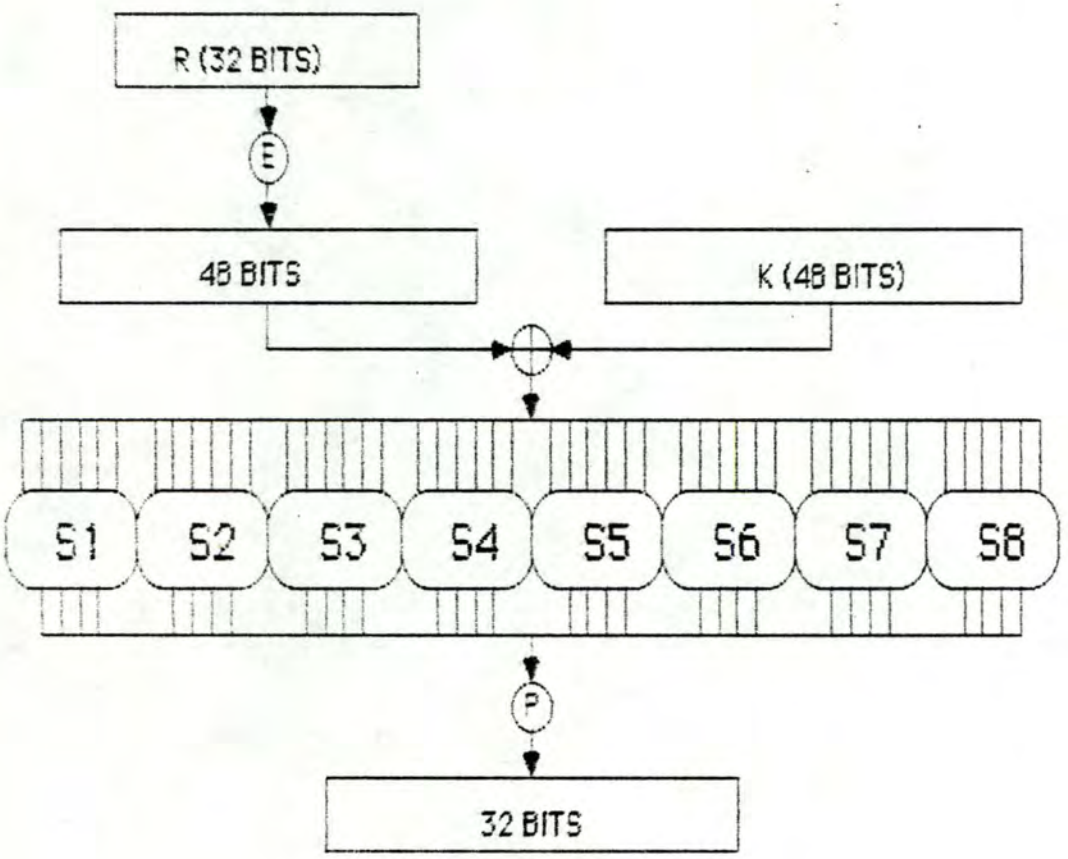


Figure 2.5 Génération de la fonction **f**

3) Algorithme général

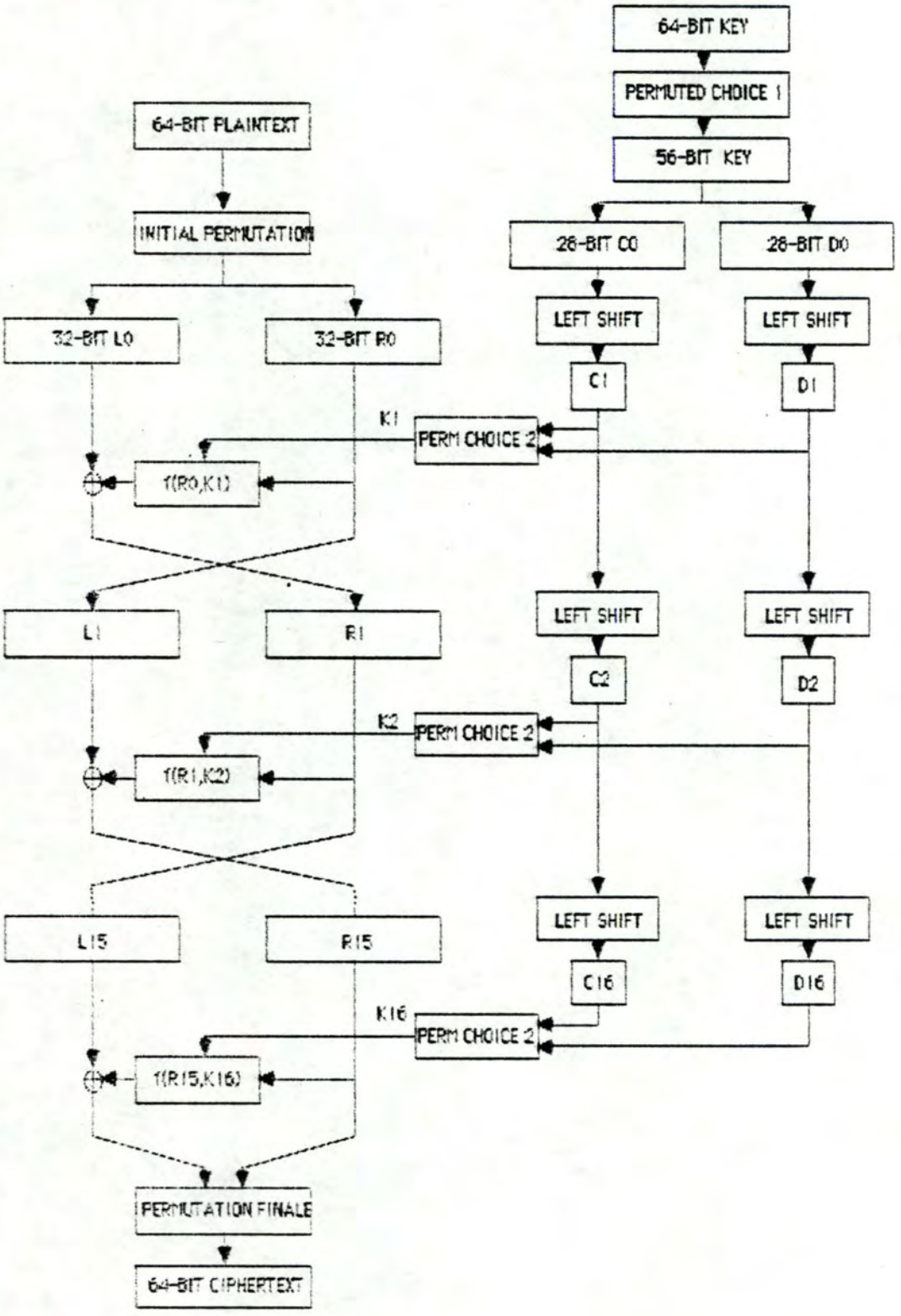


Figure 2.6 Schéma de l'algorithme général

4) Exemple d'utilisation de l'algorithme DES

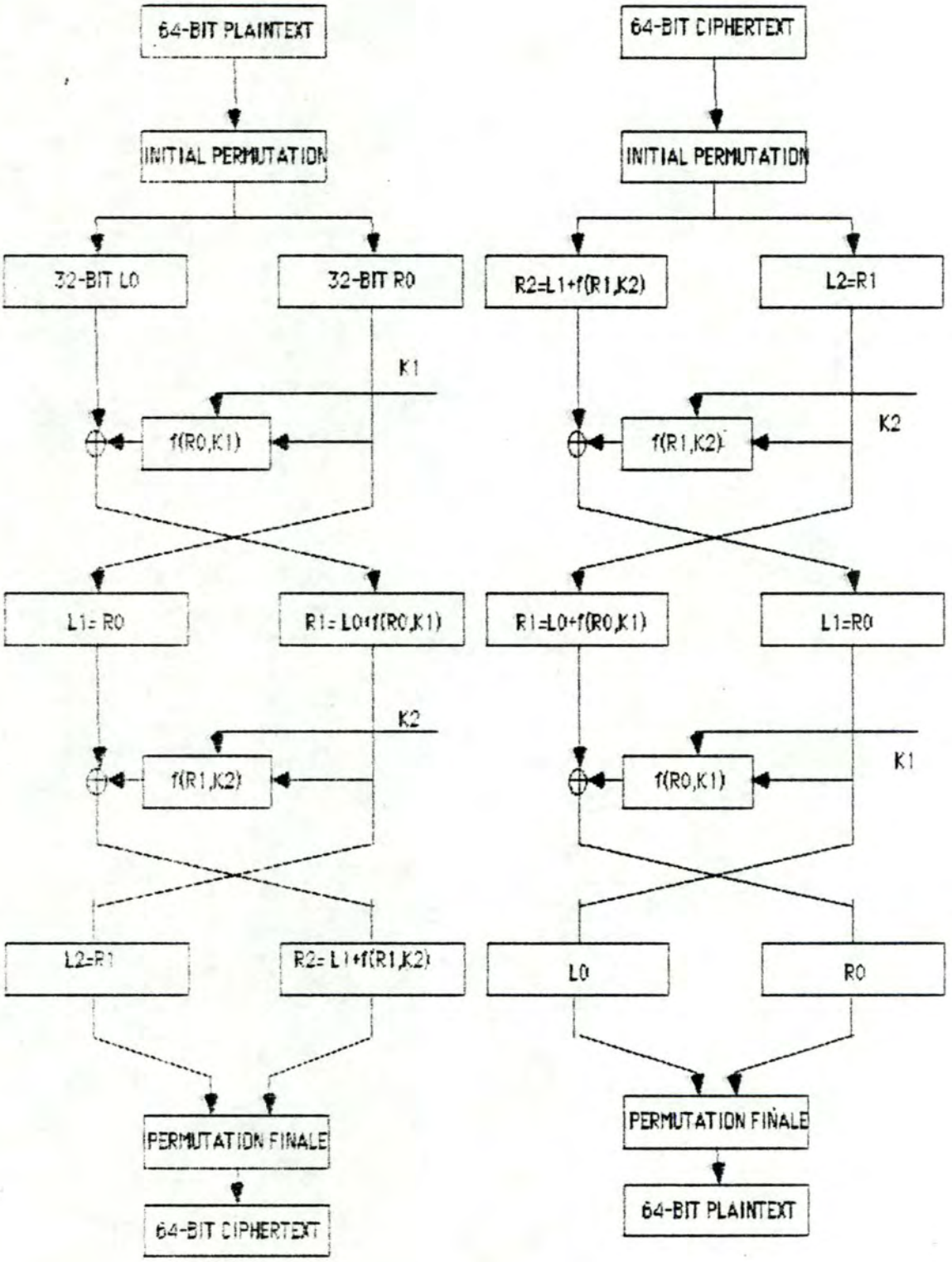


Figure 2.7 Exemple d'utilisation du DES

2.2.3 Evaluation

La sécurité d'emploi de l'algorithme DES réside dans la clé. Pour l'adversaire cherchant à découvrir la clé et disposant de plusieurs couples (texte clair, texte chiffré), il est nécessaire d'essayer les 2^{56} clés possibles jusqu'à ce que l'une d'entre elles fasse coïncider tous les couples (texte clair, texte chiffré). Si la durée d'un chiffrement est de 30 µsec, l'adversaire devra effectuer en moyenne 2^{55} chiffrements, ce qui prendrait environ 33000 ans. Il faut cependant modérer cette estimation par les diverses critiques suivantes :

1) Les principes de création et de validation du DES n'ont pas été publiés. De nombreux spécialistes prétendent qu'on ne pourra garantir sa fiabilité que lorsque ses principes de conception auront été divulgués.

2) Diffie et Hellman (1977) décrivent comment construire une machine effectuant un million de chiffrements/déchiffrements en parallèle et découvrant une clé en 12 heures en moyenne. Cette machine a été beaucoup critiquée car elle est pratiquement impossible à réaliser (problème de taille et de puissance). D'autres auteurs (Desmedt (1984)) ont étudié la question et donnent le détail d'une machine plus réaliste nécessitant 4 semaines pour trouver une clé.

3) Un autre type d'attaque décrit par Desmedt (1984) serait de trouver plusieurs clés à la fois. A première vue, si on désire trouver un million de clés, il est nécessaire d'avoir un million de couples (M_i, C_i) = (texte clair, texte chiffré) et de faire le travail pour chaque couple. Si pour tous ces couples, M_i est identique, la machine de Desmedt devrait faire le même travail pour trouver une clé que pour trouver un million de clés. Il serait alors possible de trouver en moyenne 3000 clés à l'heure (1 000 000 de clés en 4 semaines).

4) La complexité mathématique du DES est réduite si les sous-clés K_i ($1 \leq i \leq 16$) sont identiques (toutes ou par groupes). Il existe un ensemble de clés, dites clés faibles qui entraînent la génération de sous-clés identiques.

Ces clés faibles sont (ajustées avec bit de parité) :

01 01 01 01 01 01 01 01
1F 1F 1F 1F 1F 1F 1F 1F
E0 E0 E0 E0 E0 E0 E0 E0
FE FE FE FE FE FE FE FE

Pour ces clés, les registres **C** et **D** sont identiques à chaque étape. Les clés faibles ont la propriété de n'induire aucune différence entre les opérations de chiffrement et de déchiffrement, c'est-à-dire $E_k(E_k(X))$ et $D_k(D_k(X))$ sont égaux à **X**.

Il y a de plus un ensemble de clés définies comme semi-faibles qui ont la propriété de ne générer que deux sous-clés différentes. Il faut cependant remarquer que la probabilité d'obtenir une clé faible ou semi-faible à partir d'un générateur aléatoire de clés est extrêmement réduite. On peut donc ne pas tenir compte de cette possibilité ou s'assurer à titre de précaution que le nombre généré n'est pas une des 16 clés faibles ou semi-faibles, suivant l'intérêt que l'on y porte.

5) Propriété de complémentation (Davies (1981))
Si **x**, **y** et **k** sont reliés par la fonction DES de telle manière que $y = E_k(x)$ et si x', y' et k' sont les compléments à un de **x**, **y** et **k**, on peut établir que :

$$y' = E_{k'}(x')$$

Donc si on recherche la valeur de la clé **k** et qu'on connaît $Y_1 = E_k(x)$ et $Y_2 = E_k(x')$; pour chaque valeur de clé calculée **t**, on effectue $E_t(x)$ et on vérifie s'il y a égalité avec Y_1 ou Y_2. S'il y a égalité de $E_t(x)$ avec Y_1 alors $k = t$, s'il y a égalité avec Y_2 alors $k' = t'$.
En pratique, il est très rare de connaître à la fois Y_1 et Y_2.

6) On peut encore noter des régularités dans la structure des permutations et des relations entre les sorties des **S-Boxes**, mais il n'a pas été démontré que ces régularités induisaient des faiblesses dans l'algorithme.

2.2.4 Modes de réalisation

En pratique, le **DES** peut être réalisé soit par hardware soit par software.
Par hardware, le **DES** est intégré sur des chips dont les prix et les performances varient selon les constructeurs. L'implémentation par hardware présente l'avantage d'être mieux adaptée à la structure du **DES** (il a été conçu dans cette optique) et d'être extrêmement rapide (± 8 µsec pour une opération de chiffrement ou de déchiffrement). Mais les chips ne peuvent être utilisés qu'en conjonction avec des interfaces coûteux.
Par software, le **DES** est généralement programmé en langage machine pour des raisons de rapidité d'exécution. Malgré tout, si celle-ci est plus faible (30-100 msec) que par hardware, les coûts d'obtention et d'utilisation sont moins élevés (il n'est plus nécessaire de recourir à des interfaces).

Chapitre 3 Algorithme à clé publique

3.1 Introduction

L'idée de systèmes cryptographiques à clé publique est due à Diffie et Hellman (1976). L'intérêt considérable de ces algorithmes à clé publique était qu'ils permettaient de résoudre des problèmes insolubles jusque là avec les algorithmes de cryptographie classiques (transfert des clés, identification du correspondant,...)
Cependant Diffie et Hellman ne donnèrent pas dans leur article une implémentation réalisable d'algorithme à clé publique et il fallut attendre 2 ans pour que Rivest, Shamir et Adleman (1978) mettent au point le premier système à clé publique, le **RSA**.

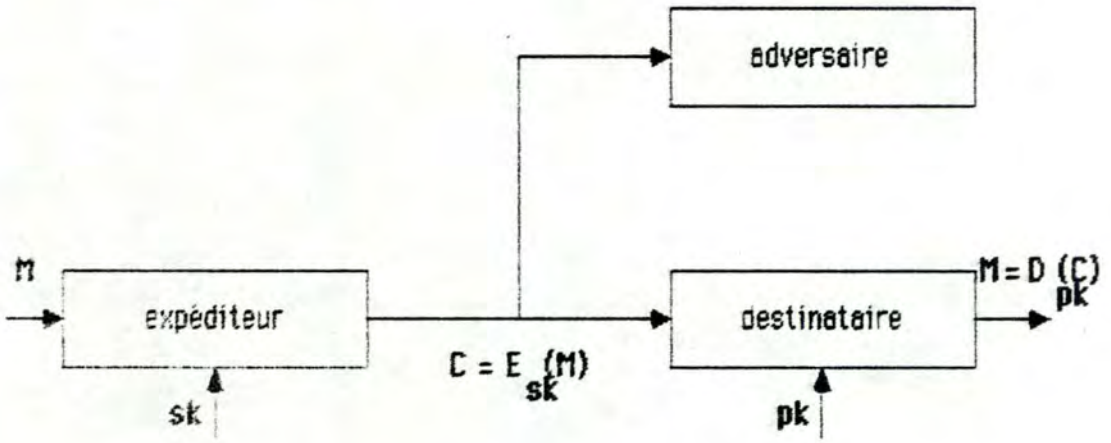


Figure 3.1 Schéma général d'échange de messages par système à clé publique

Dans un cryptosystème à clé publique, les processus de chiffrement et de déchiffrement sont calculés à partir de clés distinctes, **e** et **d** , telles que déduire **d** à partir de **e** est infaisable car prenant un temps ou un nombre d'instructions trop élevé. La clé de chiffrement **e** peut donc être révélée publiquement sans compromettre la clé de déchiffrement **d**.

Chaque utilisateur du réseau de communications recevra une clé secrète et une clé publique correspondante, il placera donc sa clé de chiffrement dans une directory publique,ainsi toute personne sera à même d'envoyer un message de telle manière que seul le destinataire sera capable de comprendre le contenu de ce message.
Grâce aux systèmes à clé publique, un échange d'informations secrètes peut avoir lieu entre deux personnes n'ayant jamais communiqué auparavant : chacune envoyant des messages chiffrés avec la clé publique de l'autre et déchiffrant les messages reçus avec sa clé secrète.
Cette propriété des systèmes à clé publique permet d'éviter entre les utilisateurs potentiels d'un réseau de télécommunications d'établir préalablement des préparatifs (échange de clés, de protocoles,...) en vue d'échanger des messages de manière sûre. Cet établissement préalable serait d'ailleurs impraticable dans un réseau de grande envergure, où un grand nombre d'utilisateurs **n**, voulant communiquer de manière secrète entre eux, résulterait en une quantité énorme d'échanges préalables (**n(n-1)/2**).
Quatre propriétés essentielles se retrouvent dans tous les systèmes à clé publique :

- déchiffrer le texte chiffré restitue le message original

$$D_d\,(E_e\,(M)) = M$$

- **E** et **D** doivent être faciles à calculer

- en révélant **e**, on doit être sûr qu'il sera impossible de déduire **d**

- si le message est d'abord déchiffré puis chiffré, on doit retrouver le texte original

$$E_e\,(\,D_d\,(M)) = M$$

3.2 L'algorithme RSA

3.2.1 Mode de calcul

L'algorithme Rivest-Shamir-Adleman (**RSA**) repose sur la difficulté de factoriser des grands nombres premiers.
Pour chiffrer un message **M**, on utilise une clé de chiffrement **(e,n)** publique (**e** et **n** sont deux entiers positifs). Le message **M** est divisé en blocs, chaque bloc m_i est représenté par un entier compris entre 0 et **(n-1)**. On élève chaque bloc m_i à la puissance **e** et on prend le reste de la division modulo **n**. Ce reste correspondra à un bloc de texte chiffré c_i.
Pour retrouver le texte clair à partir des blocs de texte chiffré c_i, on utilisera une clé de déchiffrement **(d,n)** secrète et différente de **(e,n)** (**d** et **n** sont deux entiers positifs). Il suffira d'élever chaque bloc c_i à la puissance **d** et de prendre le reste de la division modulo **n** pour retrouver m_i.
Les algorithmes de chiffrement et de déchiffrement sont donc :

$$c_i = m_i^{\,e} \pmod n \qquad \forall \text{ bloc de message } m_i \qquad (3.1)$$

$$m_i = c_i^{\,d} \pmod n \qquad \forall \text{ bloc de message chiffré } c_i \qquad (3.2)$$

Pour générer les clés de chiffrement **(e,n)** et de déchiffrement **(d,n)**, on sélectionne deux grands nombres premiers aléatoires **p** et **q**. Le nombre **n** sera le produit de **p** et de **q**. Bien que **n** soit rendu public, **p** et **q** seront gardés secrets.
On choisira **d** comme un grand nombre entier aléatoire premier avec **(p-1)*(q-1)** et plus grand que (max **(p-1,q-1)**) : il y a pour **d** une infinité de choix possibles (on prendra, par exemple, pour **d** le premier nombre premier supérieur à (max **(p-1,q-1)**).
Et enfin, on déduira **e** de la relation :

$$e.d = 1 \pmod{(p-1).(q-1)} \qquad (3.3)$$

3.2.2 Démonstration de la validité de l'algorithme **RSA**

Pour démontrer la validité de l'algorithme **RSA**, nous employerons la fonction d'Euler **Φ(p)** qui représente le nombre d'entiers positifs inférieurs à **p** et premier avec lui. Si p est un nombre premier, **Φ(p)** est égal à **p−1**.
Pour tout bloc m_i du message **M**, qui est premier avec **n**, l'équation ci-dessous dûe à Euler et Fermat établit que :

$$m_i^{\Phi(n)} = 1 \pmod n \qquad (3.4)$$

n étant obtenu comme étant le produit de **p** et de **q**

$$\Phi(n) = \Phi(p) \cdot \Phi(q) \qquad (3.5)$$
$$\Phi(n) = (p-1) \cdot (q-1) \qquad (3.6)$$

d est premier par rapport à **Φ(n)** (le plus grand commun diviseur entre **d** et **(p−1) . (q−1)** est 1)

$$e.d = 1 \pmod{\Phi(n)} \qquad (3.7)$$

Il est maintenant possible de démontrer que :

$$D(E(m_i)) = m_i \qquad (3.8)$$
$$E(D(m_i)) = m_i \qquad (3.9)$$

$$D(E(m_i)) = (E(m_i))^d = (m_i^e)^d \pmod n = m_i^{e.d} \pmod n$$
$$E(D(m_i)) = (D(m_i))^e = (m_i^d)^e \pmod n = m_i^{e.d} \pmod n$$

Par l'équation (3.7), on déduit :

$$m_i^{e.d} = m_i^{k.\Phi(n)+1} \qquad (3.10)$$

Par l'équation (3.4), si **p** ne divise pas m_i, on peut écrire :

$$m_i^{p-1} = 1 \pmod p \qquad (3.11)$$

Comme **p−1** divise **Φ(n)**, on peut écrire à partir de (3.10) et (3.11) :

$$m_i^{k.\Phi(n)+1} = m_i^{k.\Phi(n)}\ m_i = 1 \pmod p\ m_i = m_i \pmod p \quad (3.12)$$

De même, comme **q−1** divise **Φ(n)** :

$$m_i^{k.\Phi(n)+1} = m_i^{k.\Phi(n)}\ m_i = 1 \pmod q\ m_i = m_i \pmod q \quad (3.13)$$

(3.12) et (3.13) permettent de déduire :

$$m_i^{e.d} = m_i^{k.\Phi(n)+1} = m_i \pmod n \quad (3.14)$$

3.2.3 Exemple d'utilisation

Supposons que **p** = 61 et **q** = 53, **n** étant le produit de **p** et de **q**, est donc 3233.
On choisit **d** = 37.
Φ(n) est égal à **(p-1).(q-1)** donc à 3120.
On calcule **e** par la formule (3.3) ce qui donne **e** = 253.
Chiffrons le mot "CIPHER". On affecte à chaque lettre une valeur numérique correspondant à la place que cette lettre occupe dans l'alphabet et on groupe les lettres par deux. Les valeurs numériques de chaque bloc de deux lettres iront de 0000 (blanc-blanc) à 2626 (ZZ) et seront donc toutes comprises entre **0** et **n**.

Le premier bloc m_1 sera donc 0309 (CI).

Le second bloc m_2 sera 1609 (PH).

Le troisième bloc m_3 sera 0518 (ER).

Le chiffrement de m_1 se fera de la manière suivante :

$$309^{253} = 1971 \pmod{3233}$$

Le premier bloc de texte chiffré c_1 sera donc 1971.

Pour retouver m_1, il suffit de faire :

$$1971^{37} = 309 \pmod{3233}$$

3.2.4 Evaluation

On peut évaluer l'algorithme **RSA** en estimant les temps nécessaires à la réalisation des différents modes d'attaque possibles.
Le premier de ces modes d'attaque est d'essayer de factoriser **n**, ce qui permettrait de retrouver **Φ(n)**. Il existe un grand nombre d'algorithmes de factorisation, le temps nécessaire à la factorisation de **n** dépendera donc de l'algorithme employé mais aussi de la taille de **n**. Ainsi Rivest, Shamir et Adleman ont calculé les temps de factorisation de **n** avec l'algorithme de factorisation le plus rapide connu.

nombre de chiffres de n	nombre d'opérations à effectuer	temps si une opération = 1 µsec
50	$1,4 . 10^{10}$	3,9 heures
75	$9,0 . 10^{12}$	104 jours
100	$2,3 . 10^{15}$	74 ans
200	$1,2 . 10^{23}$	$3,8 . 10^{9}$ années
300	$1,5 . 10^{29}$	$4,9 . 10^{15}$ années
500	$1,3 . 10^{39}$	$4,2 . 10^{25}$ années

Tableau 3.1 Evaluation du temps de factorisation de **n**

Un autre mode d'attaque serait d'essayer de déduire **Φ(n)** sans calculer **n**, afin de calculer la clé secrète **d** comme étant l'inverse de **e** modulo **Φ(n)**. Ce mode d'attaque est en fait aussi complexe que celui consistant à factoriser **n**.
Un troisième mode d'attaque viserait à déterminer **d** sans factoriser **n** ou déduire **Φ(n)**. Cela reviendrait à énumérer toutes les valeurs possibles pour **d**. Ce mode d'attaque est le plus long à réaliser pour autant que la valeur de **d** qui a été choisie, soit très grande.
Dans tous les cas, si on utilise des nombres **p** et **q** de l'ordre de cent chiffres, on pourra être certain que toute factorisation ou énumération avec les algorithmes actuels est impossible dans des temps raisonnables.

3.2.5 Réalisation des différentes étapes

1) Chiffrement/Déchiffrement

Pour réaliser l'élevation de chaque bloc $\mathbf{m_i}$ à la puissance **e** et obtenir le reste de la division modulo **n**, on utilise un algorithme appelé "exponentiation par élévation au carré et multiplication".

Etape 1 Transformer **e** en sa représentation binaire $e_k e_{k-1} \ldots e_1 e_0$

Etape 2 Mettre $\mathbf{c_i}$ à 1

Etape 3 Répéter 3a et 3b pour j = k, k-1, ..., 1,0

Etape 3a $\mathbf{c_i} = \mathbf{c_i}.\mathbf{c_i} \pmod{\mathbf{n}}$

Etape 3b Si $e_j = 1$ alors $\mathbf{c_i} = \mathbf{c_i}.\mathbf{m_i} \pmod{\mathbf{n}}$

Etape 4 Stop.$\mathbf{c_i}$ est la forme chiffrée de $\mathbf{m_i}$.

2) Génération de grands nombres premiers

Pour trouver un nombre premier de cent chiffres, on génère des nombres impairs aléatoires de cent chiffres jusqu'à ce qu'un nombre premier soit trouvé.
Pour vérifier qu'un nombre **b** est un nombre premier, on utilisera un algorithme probabiliste. Cet algorithme génère un entier aléatoire **a** compris entre **1** et **b-1** et teste si :

le plus grand diviseur entre **a** et **b** est égal à **1**

$$\mathbf{J(a,b) = (a^{b-1})(mod\ b)} \qquad (3.15)$$

J(a,b) est le symbole de Jacobi. Si **b** est premier, la relation (3.15) est toujours vraie. Si **b** est factorisable, la relation (3.15) aura une chance sur deux d'être fausse.
Pour cent valeurs aléatoires de **a**, si la relation est vraie, il ne subsiste plus qu'une chance négligeable ($1/2^{100}$) pour que **b** soit factorisable.

3) Choix de **d**

Pour choisir **d**, n'importe quel nombre premier plus grand que **p** et **q** fera l'affaire. Le nombre **d** doit être choisi le plus grand possible pour

rendre infaisable toute attaque énumérative.

4) Réduction de **e** à partir de **d** et de $\Phi(n)$

Pour déduire **e**, on utilise une variation de l'algorithme d'Euclide qui calcule le plus grand commun diviseur de $\Phi(n)$ et de **d**.
On calcule donc le pgcd $(\Phi(n),d)$ avec une série $x_0,x_1,x_2,$ ... telle que :

$$x_0 = \Phi(n)$$
$$x_1 = d$$
.
.
.
$$x_{i+1} = x_{i-1} \pmod{x_i}$$

jusqu'à ce qu'on obtienne $x_k = 0$.
On calcule ensuite pour chaque x_i, les nombres a_i et b_i tels que :

$$x_i = a_i \cdot x_0 + b_i \cdot x_1$$

Si $x_{k-1} = 1$ alors b_{k-1} est l'inverse multiplicatif de $x_1 \pmod{x_0}$.

Chapitre 4 Signatures électroniques

4.1 Introduction

Avec la rapidité de la marche des affaires et les distances considérables entre contractants, le temps nécessaire pour l'obtention d'une signature manuscrite peut retarder exagérément la conclusion d'un projet. C'est pourquoi il serait intéressant de disposer d'un système de communications permettant de valider les messages par le biais de signatures digitales. Il suffirait de respecter certaines contraintes pour que la substitution des signatures manuscrites par les signatures électroniques soit efficace :

a) le destinataire du message devrait pouvoir utiliser la signature comme moyen de preuve pour identifier l'expéditeur et authentifier le contenu du message.

b) l'expéditeur devrait être certain qu'il sera impossible de créer de faux messages portant sa signature ou de falsifier le contenu d'un message qu'il aurait pu envoyer.

Une signature électronique sera une série de bits générée en fonction :

a) du message

b) d'informations propres à l'expéditeur et qui ne sont connues par personne d'autre (clé secrète).

c) d'informations disponibles à la fois à l'expéditeur et au destinataire (algorithmes, paramètres de validation,.....).

Les informations permettant de générer une signature digitale doivent être suffisantes pour pouvoir la valider mais insuffisantes pour pouvoir la falsifier, c'est pourquoi la procédure de génération et celle de la validation doivent être différentes.
Ainsi dans le réseau de communication, l'expéditeur **A** pourra envoyer des messages signés au destinataire **B**, en utilisant des procédures prédéfinies : **A** devra avoir l'information qui lui permettra de générer une signature pour chaque message transmis à **B** et **B** l'information nécessaire pour valider les messages et les signatures reçus de **A**.
Diverses techniques cryptographiques peuvent être utilisées pour générer des signatures digitales. On utilisera les algorithmes à clé secrète ou à clé publique suivant le type de signature que l'on désire implémenter.
On distingue deux types de signatures électroniques :

a) les signatures dites ARBITREES qui sont validées par un "arbitre" (souvent un nœud dans le réseau de communications) au moment de la réception d'un message. Elles sont implémentées de préférence par algorithme à clé secrète. De telles signatures sont appropriées pour un ensemble d'utilisateurs appartenant à la même organisation et qui se conformeraient aux décisions de l'arbitre.

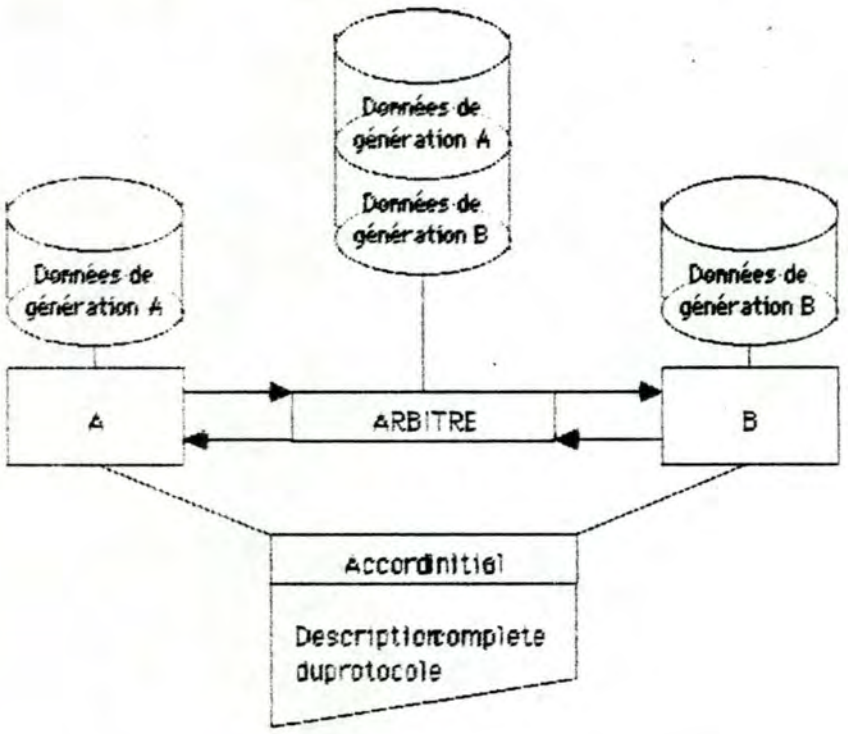


Figure 4.1 Schéma d'un système à signatures arbitrées

b) les signatures dites VRAIES ou UNIVERSELLES qui peuvent être validées par toute personne ayant accès aux paramètres publics de validation. Elles sont implémentées de préférence par algorithme à clé publique. Les signatures vraies sont plus souples, car elles peuvent être utilisées dans les cas où il serait impossible d'avoir recours à un arbitre, par exemple dans le cas d'utilisateurs ayant des intérêts opposés.

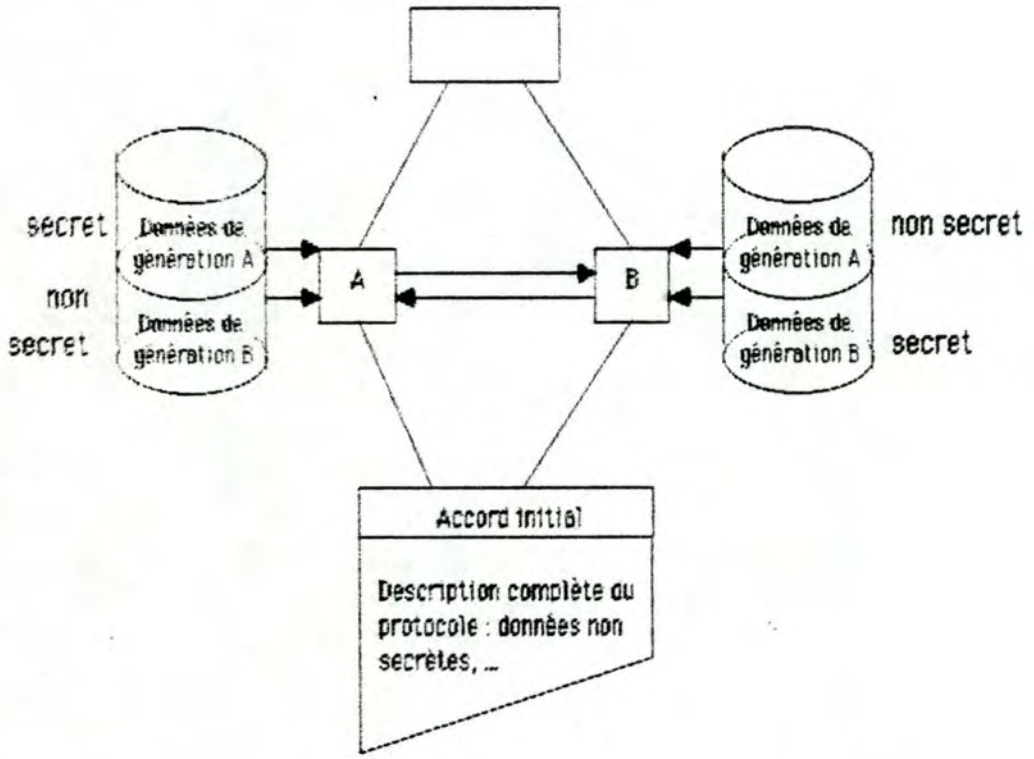


Figure 4.2 Schéma d'un système à signatures universelles

4.2 Signatures arbitrées

4.2.1 Introduction

Dans un système de communications à signature arbitrée (appelée aussi code d'authentification), les messages signés sont envoyés à un arbitre, qui partage avec les expéditeurs potentiels du système les méthodes employées pour la génération des signatures. L'arbitre authentifiera les messages et signatures en provenance de chaque utilisateur. Toute falsification du message ou de la signature est impossible car les informations de génération de la signature ne sont pas connues par les autres utilisateurs du système.
Il est essentiel que deux messages n'aient pas le même texte car la technique de génération fait que la même signature est obtenue si on utilise les mêmes paramètres de départ (clé, message). Il faut donc ajouter différentes rubriques au texte des messages pour éviter qu'on puisse connaître à l'avance leurs signatures :

- <Identifiant du message>
- <Date, heure>

La technique d'authentification des messages est donc une procédure permettant d'identifier l'expéditeur et de certifier le contenu. Il existe diverses procédures d'authentification de messages, la plus courante se base sur l'algorithme **DES**. Le choix de telle ou telle procédure dépend de la volonté des utilisateurs du système de préserver ou non la confidentialité des messages échangés.

4.2.2 Signatures arbitrées avec confidentialité des messages

Le **DES** transforme 64 bits de message en clair (**X**) en 64 bits de message chiffré (**Y**). Pour cette transformation, le **DES** utilise une clé cryptographique de 56 bits (**k**). Chaque utilisateur du système **A, B, .., Z** possède sa propre clé secrète k_a, k_b, .., k_z connues également par l'arbitre, ainsi celui-ci pourra authentifier les messages en provenance de chaque utilisateur et chaque utilisateur pourra certifier les messages en provenance de l'arbitre.
Pour générer un code d'authentification, on procède de la manière suivante :

- Diviser le message **M** en **n** blocs de 64 bits : $M = (X_1, X_2, X_3, .., X_n)$
- Ajouter en fin de message un bloc de 64 bits, X_{n+1} préalablement initialisé à zéro, ce bloc servira à la génération du code d'authentification du message.

- Chiffrer les blocs $(X_1, X_2, X_3, .., X_n, X_{n+1})$ selon la technique suivante:

$$Y_1 = E_k (X_1 \oplus V)$$

$$Y_2 = E_k (X_2 \oplus Y_1 \oplus X_1)$$

$$Y_n = E_k (X_n \oplus Y_{n-1} \oplus X_{n-1})$$

$$Y_{n+1} = E_k (X_{n+1} \oplus Y_n \oplus X_n)$$

où $\oplus$ représente l'opération d'addition modulo 2, **k** la clé secrète de l'expéditeur et **V** un vecteur d'initialisation aléatoire non secret.

- Envoyer au destinataire les informations $(V, (Y_1, Y_2, .., Y_n), Y_{n+1})$
- Le bloc de 64 bits Y_{n+1} représente le code d'authentification du message.

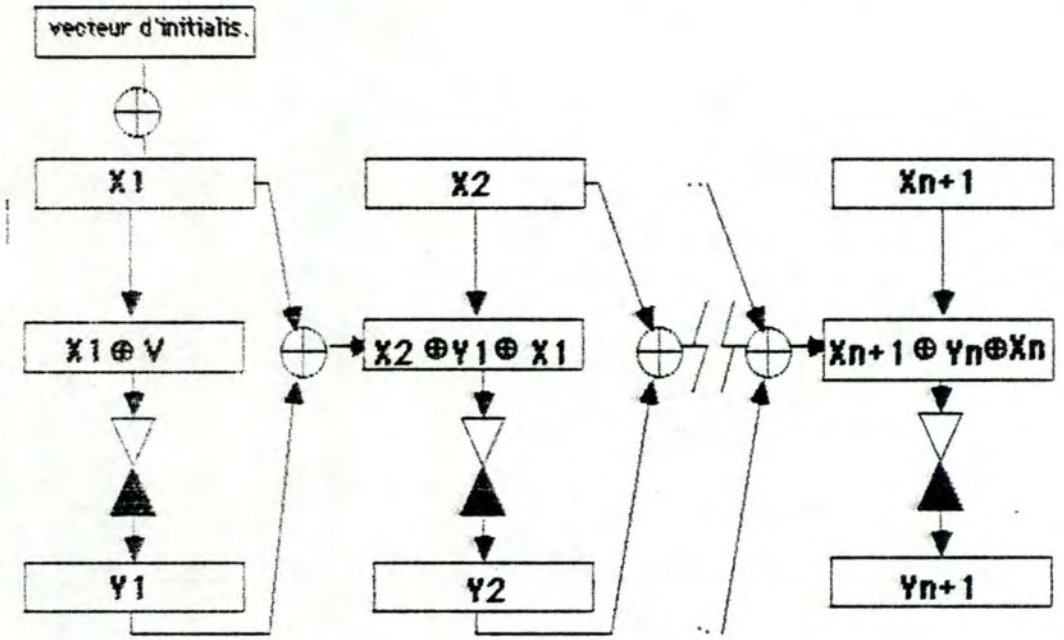


Figure 4.1 Chiffrement d'un message avec génération de signature arbitrée

- Pour récupérer le message initial, le destinataire (ici l'arbitre) devra effectuer les opérations ci-dessous :

$$X_1 = D_k(Y_1) \oplus V$$

$$X_2 = D_k(Y_2) \oplus Y_1 \oplus X_1$$

$$X_n = D_k(Y_n) \oplus Y_{n-1} \oplus X_{n-1}$$

$$X_{n+1} = D_k(Y_{n+1}) \oplus Y_n \oplus X_n$$

- le destinataire vérifiera ensuite que le bloc X_{n+1} trouvé est bien égal à zéro. Si c'est le cas, le contenu du message est certifié correct. Comme X_{n+1} est une fonction complexe de la clé **k**, du vecteur d'initialisation **V** et du texte chiffré $(Y_1, Y_2, ..., Y_n, Y_{n+1})$, toute modification de l'un de ces paramètres aura pour effet de rendre la valeur de X_{n+1} différente de zéro.

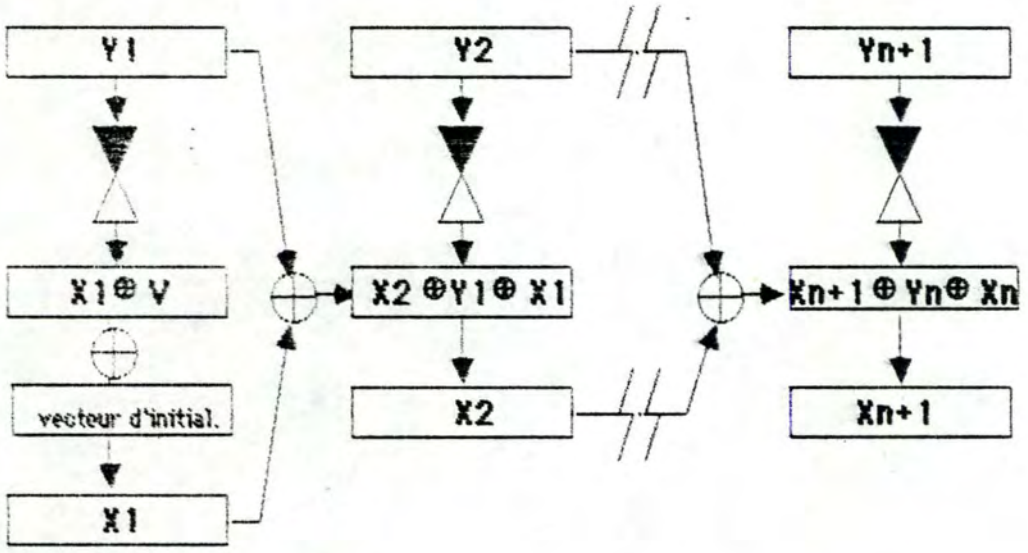


Figure 4.2 Déchiffrement d'un message porteur d'une signature arbitrée

4.2.3 Signature arbitrée sans confidentialité des messages.

Si la confidentialité du message n'est pas nécessaire, les opérations décrites ci-dessus peuvent être simplifiées. Le **DES** sera toujours l'algorithme utilisé pour générer la signature du message.
Pour générer un code d'authentification, on procède de la manière suivante :

- Diviser le message **M** en **n** blocs de 64 bits : $M = (X_1, X_2, X_3, .., X_n)$
- Chiffrer le premier bloc X_1 avec le **DES** et une clé secrète **k**.
- Additionner modulo 2 les 64 bits suivants du message avec les 64 bits résultant du chiffrement ci-dessus.
- Chiffrer le résultat de l'addition avec le **DES** et la même clé secrète et additionner modulo 2 avec les 64 bits suivants du message.
- Continuer ainsi jusqu'à ce que tout le message ait été pris en compte. Le code d'authentification du message sera les 64 bits du résultat final.
- Envoyer le message **M** auquel on ajoute les 64 bits d'authentification.
- Lors de la réception du message, le destinataire (l'arbitre) vérifiera par le même schéma que le message reçu fournit les mêmes 64 bits d'authentification que ceux envoyés. Si c'est le cas, le contenu du message est certifié correct.

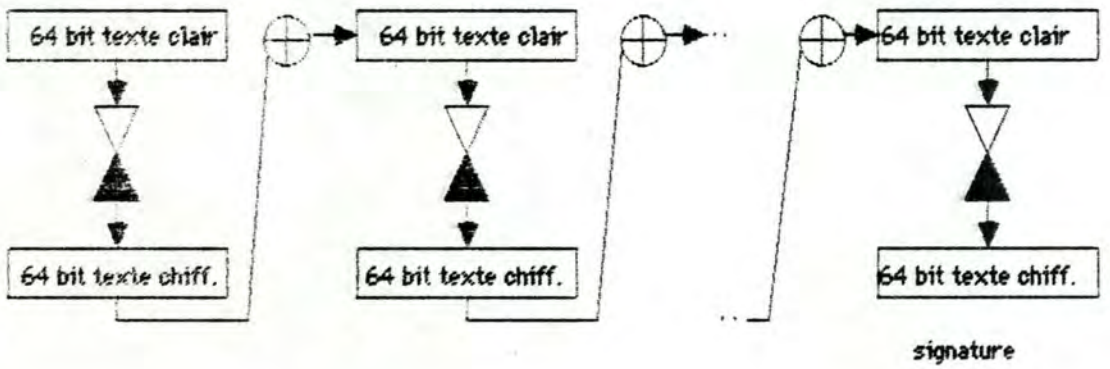


Figure 4.3 Signature arbitrée sans confidentialité

4.2.4 Conclusion

Ces deux méthodes permettent de détecter toute modification du contenu du message, car une modification même minime aura des effets importants dans le calcul du code d'authentification. La deuxième méthode est parfois plus intéressante que la première pour certaines applications (bancaires par exemple) étant donné que le message reste compréhensible à tout moment.
Il faut cependant tenir compte du fait que la fraude par duplication ou la perte des messages n'est prise en charge que si on n'omet pas d'inclure et de vérifier les informations identifiant le message (ainsi si on découvre deux messages présentant des informations exactement semblables, on saura qu'il y a eu duplication).

4.3 Signatures universelles

4.3.1 Introduction

Dans le cas de signatures universelles, le destinataire peut lui-même valider chaque message et signature sans avoir recours à un arbitre.
Pour éviter la falsification des signatures et garantir leur provenance, chaque utilisateur possède une série d'informations qu'il ne partage pas avec ses correspondants et qui lui sert à générer sa signature personnelle. Le message et sa signature seront vérifiés par le destinataire à partir d'informations publiques sur lesquelles les correspondants se seront au préalable mis d'accord.

4.3.2 Signatures universelles par algorithme à clé publique

Pour générer des signatures universelles avec un algorithme à clé publique, il suffit de donner à chaque utilisateur **A**, **B**, **C**, ..., **Z** une clé publique P_a, P_b, P_c, ..., P_z et une clé secrète S_a, S_b, S_c, ..., S_z . Ces clés sont telles que, pour un message **M** quelconque :

$$E_{Pi}\,(\,D_{Si}\,(M)\,) = M \qquad \forall i : a \le i \le z$$

$$E_{Si}\,(\,D_{Pi}\,(M)\,) = M \qquad \forall i : a \le i \le z$$

La signature d'un message sera générée simplement en chiffrant le message **M** avec la clé secrète de l'expéditeur $E_{Si}\,(M) = C$. Le message pourra être ensuite envoyé tel quel au destinataire qui le déchiffrera au moyen de la clé publique de l'expéditeur $D_{Pi}\,(C) = M$. Si on transmet le message de la sorte, il faut prendre garde au fait que la confidentialité ne sera pas préservée ; en effet pour rendre le message intelligible, il suffit de le transformer avec la clé publique de l'expéditeur. Cette clé étant publique, est donc accessible à tous.
Si on désire préserver la confidentialité du message, l'expéditeur le chiffrera d'abord avec sa clé secrète puis il chiffrera le résultat avec la clé publique de son destinataire $E_{Pk}\,(E_{Si}\,(M)) = C$. Ainsi seul le destinataire sera à même de rendre le message compréhensible en le transformant d'abord avec sa clé secrète personnelle puis avec la clé publique de l'expéditeur $D_{Pi}\,(D_{Sk}\,(C)) = M$.
L'authentification de l'expéditeur est assurée par le fait que celui-ci est le

seul à posséder la clé secrète : donc il est le seul à avoir pu produire ce message chiffré.

4.3.3 Signatures universelles par algorithme à clé secrète

Lamport a mis au point une méthode basée sur le **DES** pour générer des signatures universelles. Lorsqu'on désire expédier un message de **n** bits, on génère **2 * n** clés secrètes de 56 bits

$$k_1, K_1, k_2, K_2, \ldots, k_n, K_n$$

et on envoie au destinataire deux groupes de **2 * n** paramètres de validation

$$u_1, U_1, u_2, U_2, \ldots, u_n, U_n$$

$$v_1, V_1, v_2, V_2, \ldots, v_n, V_n$$

tels que $\forall\ 1 <= i <= n$: $v_i = E\ k_i\ (u_i)$ et $V_i = E\ K_i\ (U_i)$

Suivant que le premier bit du message est 0 ou 1, on utilisera k_1 ou K_1 respectivement pour générer la signature. Suivant que le deuxième bit du message est 0 ou 1, on utilisera k_2 ou K_2 pour générer la suite de la signature et ainsi de suite pour les **n** bits du message. La signature globale sera les **n * 56** bits des clés.
Lors de la réception du message, si le premier bit reçu est 0, le destinataire vérifiera que les 56 premiers bits de la signature transforment bien u_1 en v_1. Si le premier bit reçu est 1, alors il faut que les 56 premiers bits de la signature transforment U_1 en V_1. On procède de même pour les bits suivants du message.

Cette méthode n'est pas très efficace car les paramètres de validation ne peuvent servir qu'une fois et que la signature peut atteindre des proportions importantes (56 fois plus longue que le message).
L'efficacité peut être améliorée par diverses techniques de compression :

1) Divisons le message en blocs de 56 bits $X_1, X_2, ..., X_n$ et utilisons ces blocs comme des clés cryptographiques. On définit la compression du message **M** comme étant :

$$CE(M) = E_{X_n}(E_{X_{n-1}}(... (E_{X_2}(E_{X_1}(c))) ...))$$ où **c** est une constante publique

La signature sera réalisée comme cela a été décrit ci-dessus mais en se basant non plus sur le message proprement dit mais sur sa version compressée **CE (M)** . Comme **CE (M)** contient 64 bits, la signature du message sera longue de **56 * 64** bits c'est-à-dire **3584** bits au lieu de **56 * n** bits auparavant. Si **n** était un nombre élevé, un gain important peut être réalisé.
Il faut prendre garde au fait que des messages différents pourraient avoir la même version compressée, bien qu'il soit pratiquement impossible de trouver **M'** tel que **CE (M) = CE (M')** même si **M** et **CE (M)** sont connus.
2) Il est aussi possible de réduire la longueur de la signature en ne considérant que les **p** premiers bits du message compressé ($p < 64$).
La probabilité, que les **p** premiers bits de **CE (M)** soient égaux aux **p** premiers bits de **CE (M')** si **M'** est différent de **M**, est égale à $1/2^p$. On déterminerait une bonne valeur de **p** en s'arrangeant pour que le nombre moyen d'essais, qu'un adversaire devrait réaliser pour trouver un **M'** ayant la même signature que **M**, soit le plus élevé possible (ce nombre moyen est bien entendu égal à $1/2^{p-1}$).

Il existe encore d'autres méthodes plus complexes mais qu'il serait fastidieux d'énumérer ici. Le lecteur intéressé pourra trouver dans Meyer et Matyas (1982) un exposé détaillé de l'ensemble des méthodes de génération de signatures universelles par algorithme à clé secrète.

4.4 Critique des différents types de signatures

4.4.1 Signatures par algorithmes à clé publique

La vitesse d'exécution est peu élevée. Les vitesses atteintes vont de 10000 bits/sec à 1 Mbits/sec pour l'algorithme **RSA**. Ceci est dû au fait qu'à l'heure actuelle, il n'existe pas encore de réalisation d'un système à clé publique par hardware.

La perte ou le vol des clés secrètes est un événement grave car en cas de perte de la clé, il n'y a aucun moyen de la récupérer et en cas de vol, tous les messages signés par le voleur ne pourront être distingués des messages authentiques signés par le propriétaire de la clé. De plus, il sera nécessaire de prévenir tous les correspondants car tous les messages adressés au propriétaire de la clé pourront être compris par le voleur.

L'authenticité des clés publiques doit être garantie sinon le message qu'on désire envoyer, pourrait être chiffré avec la clé publique d'un adversaire et non pas avec celle du correspondant.

Les algorithmes à clé publique ont pour la plupart, été brisés (**Knapsack, RSA** pour certaines clés, ...)

4.4.2 Signatures par algorithmes à clé secrète

Le transfert des clés doit se faire par des moyens très sûrs. Si la clé peut être connue d'un adversaire, tous les messages échangés pourront être signés par celui-ci.
La vitesse d'exécution dépend de l'implémentation utilisée : hardware ou software. Elle est plus rapide que dans le cas des algorithmes à clé publique.
En cas de perte de clé, la situation est moins catastrophique que pour un algorithme à clé publique car le correspondant possède toujours sa clé qui est la même que celle de l'expéditeur. En cas de vol, seul le correspondant partageant la clé avec le propriétaire légitime doit être prévenu.
Diverses publications sur les faiblesses des algorithmes à clé secrète (Meyer et Matyas (1982), Desmedt (1984), Diffie et Hellman (1977), Beker et Piper (1982), ...) établissent qu'aucun d'entre eux ne peut être considéré comme absolument sûr. Cependant différentes techniques simples peuvent être mises en œuvre pour garantir plus de fiabilité. Ainsi, par exemple, l'allongement des clés de chiffrement (clés de 128 ou de 256 bits au lieu des 56 bits pour les clés du **DES**) ou la génération du code

d'authentification par la méthode de "triple chiffrement". Cette méthode diffère de celle décrite au paragraphe 4.2.1 par l'utilisation de deux clés secrètes k_1 et k_2. Pour calculer le code d'authentification, on procède comme auparavant en utilisant k_1 à la place de k comme clé de chiffrement. Lorsque le dernier bloc du message a été chiffré avec k_1, on utilise k_2 pour déchiffrer le résultat, puis on chiffre le résultat du déchiffrement à nouveau avec k_1. Cette méthode rend plus complexe le code d'authentification et augmente la difficulté pour un adversaire de découvrir les clés permettant de générer la signature.

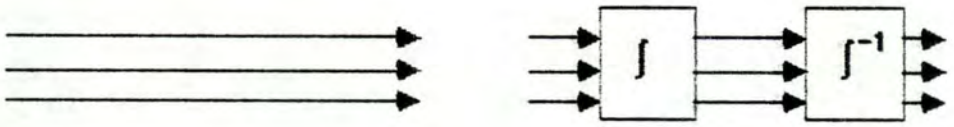
Chapitre 5 Alternatives à la présentation NBS du DES : optimisations possibles

5.1 Introduction

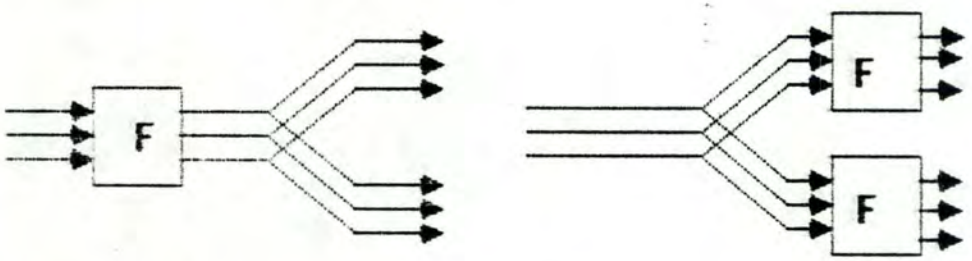
La description du DES donnée dans le document du National Bureau of Standards est peut-être la plus mauvaise manière de présenter le DES d'un point de vue software. On peut dès lors effectuer des transformations de la structure du DES sans en changer le comportement entrée-sortie de façon à obtenir une version plus simple et plus efficace.
Nous verrons d'abord quelques transformations élémentaires qui permettront de simplifier le schéma de l'algorithme DES de la figure 2.6 et puis nous appliquerons ces transformations pour obtenir des améliorations notables.

5.2 Transformations élémentaires

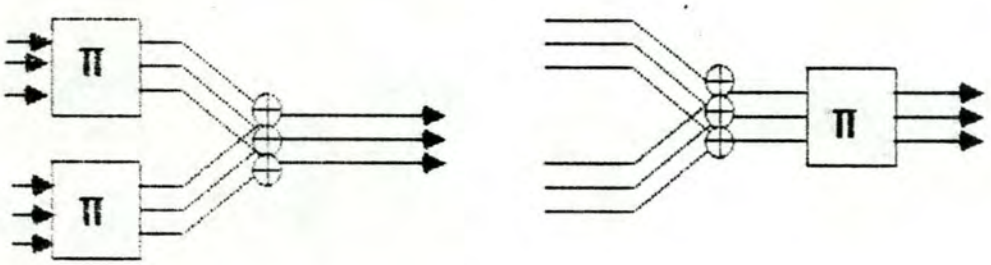
a) On peut insérer dans le schéma du DES, à n'importe quel endroit de celui-ci, une permutation à condition qu'elle soit suivie par la permutation inverse correspondante. Inversément, on peut supprimer du schéma, une permutation qui serait suivie de son inverse



b) On peut propager une permutation au-delà d'un point de divergence dans le schéma DES, et inversément.



c) On peut déplacer deux permutations identiques au-delà d'un opérateur tel que l'opérateur d'addition modulo deux, et inversément.



5.3 Application de ces transformations élémentaires.

5.3.1 Gain de la table P

Aux points A et B de la figure 5.1.1, on insère une permutation P et son inverse P^{-1} (transformation a). La permutation P au point A peut maintenant se propager au-delà du XOR (transformation c) et venir annuler la permutation P^{-1} au point B suivant (inverse de la transformation a). La permutation au point B peut se propager vers la fonction d'expansion E et vers la permutation P^{-1} au point A suivant (transformation b). Cela nous donnera la figure 5.1.2. Cette transformation est intéressante : par la combinaison des permutations E et P, on peut réduire le nombre de tables sans augmenter la dimension des tables restantes.

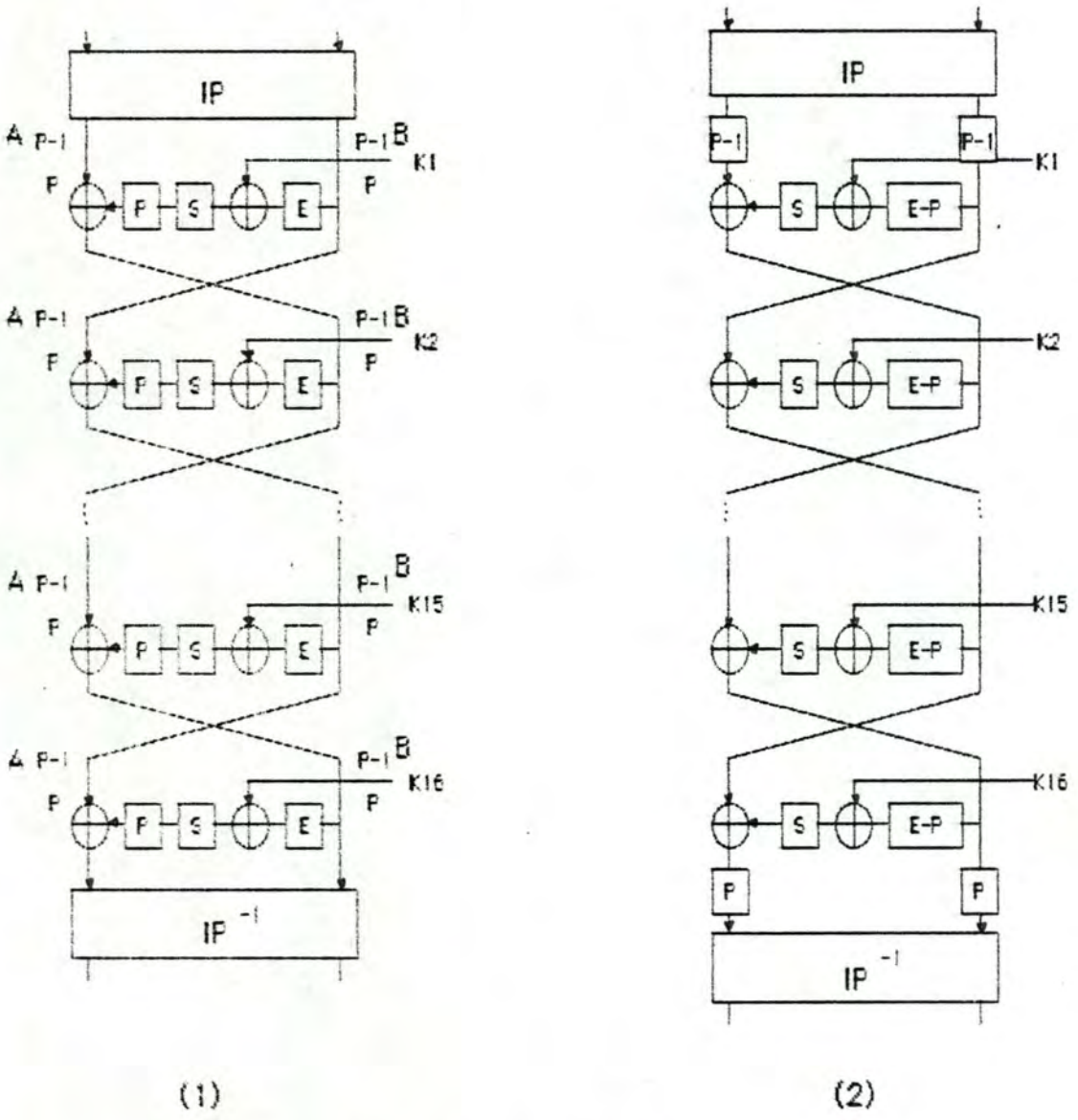


Figure 5.1 Gain de la table **P**

5.3.2 Modèle d'un DES à 48 bits

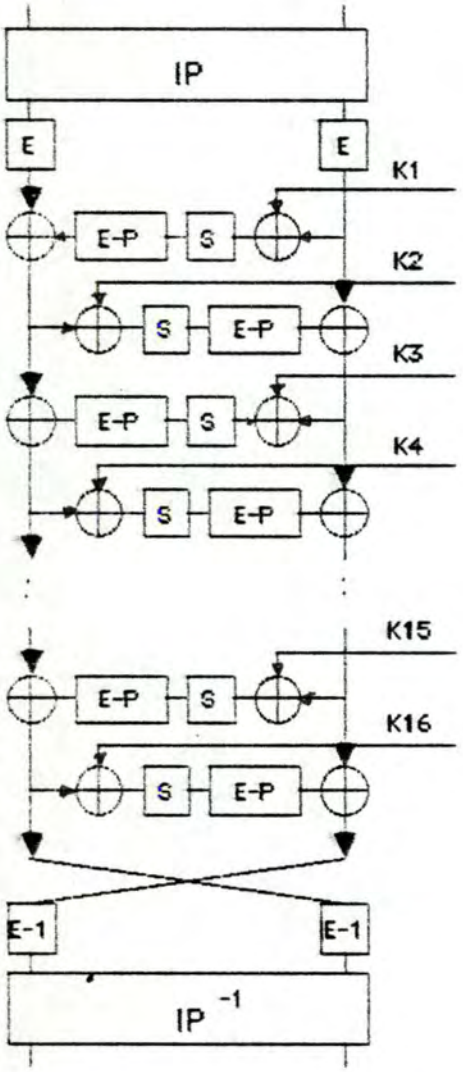


Figure 5.2 Modèle DES à 48 bits

En partant du schéma 5.1.1, on peut, au lieu de placer P et P^{-1} aux points A et B, placer E et E^{-1}. Par un raisonnement analogue à celui fait ci-dessus, on peut simplifier le diagramme et obtenir ainsi la figure 5.2.
Dans ce diagramme, on remarque que les liaisons entre les différentes permutations véhiculent 48 bits au lieu de 32 auparavant (E a en sortie 48 bits et la combinaison E-P également).
Ce modèle est particulièrement intéressant pour les processeurs 48 bits et il est également remarquable par le fait qu'il ne nécessite plus que 8 itérations à deux étapes au lieu de 16.

5.4 Autres transformations

5.4.1 Version itérative du DES

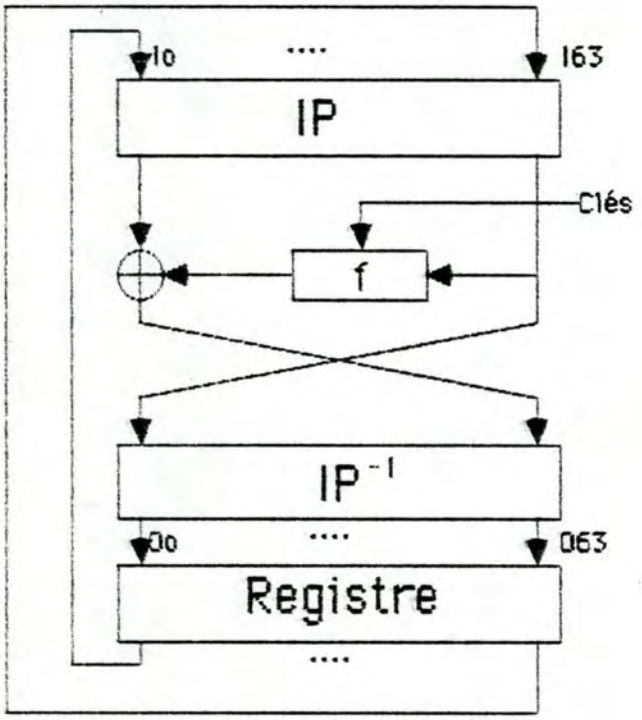


Figure 5.3 Version itérative du DES

On peut, pour obtenir une version itérative du DES, omettre les permutations initiales et finales (respectivement **IP** et $\mathbf{IP}^{-1}$) en début et en fin d'algorithme, et les placer juste avant les registres gauche et droit lors des itérations. On obtient alors le schéma purement itératif de la figure 5.3. Le DES obtenu est composé de 16 itérations identiques (sauf la dernière qui ne nécessite pas de swapping).
Appelons $I_0, I_1, I_2, ..., I_{63}$ les 64 bits d'entrée à la permutation initiale **IP** et $O_0, O_1, O_2, ..., O_{63}$ les 64 bits de sortie de la permutation finale $\mathbf{IP}^{-1}$, il est possible de dériver des relations entre les bits d'entrée et les bits de sortie .

Pour tout $0 \leq j \leq 31$: $O_{2j} = I_{2j+1} + Yk_j$

$$O_{2j+1} = I_{2j}$$

5.4.2 Utilisation des caractéristiques analytiques de certaines permutations

Divers composants du DES présentent des régularités numériques qui suggèrent l'existence de représentations analytiques. L'avantage de telles représentations est de pouvoir exécuter une permutation sur base d'une équation plutôt que d'une table.

a) La permutation initiale **IP**

La permutation **IP** permute les 64 bits d'entrée $X_0, X_1, X_2, ..., X_{63}$ et donne en sortie les bits $Y_1, Y_2, Y_3, ..., Y_{63}$. En associant aux index de ces bits leur représentation binaire, on peut observer des relations entre les bits d'entrée et les bits de sortie.
Soit X_i (bit d'entrée d'indice i) et Y_j (bit de sortie d'indice j)
i étant représenté par le nombre binaire $(x_5, x_4, x_3, x_2, x_1, x_0)$ tel que

$$i = \sum_{k=0}^{5} x_k 2^k$$

j étant représenté par le nombre binaire $(y_5, y_4, y_3, y_2, y_1, y_0)$ tel que

$$j = \sum_{l=0}^{5} y_l 2^l$$

Dans la permutation **IP**, on a la relation suivante :

$$(y_5, y_4, y_3, y_2, y_1, y_0) = (x_0, x_2, x_1, x_5, x_4, x_3)$$

Dans la permutation $\mathbf{IP}^{-1}$, on a la relation suivante :

$$(x_5, x_4, x_3, x_2, x_1, x_0) = (y_2, y_1, y_0, y_4, y_3, y_5)$$

b) La permutation d'expansion **E**

Dans la phase de permutation **E**, 32 bits d'entrée numérotés $r_0, r_1, r_2, \ldots, r_{31}$ sont permutés et donnent 48 bits comme résultat ($v_0, v_1, v_2, \ldots, v_{47}$) . On peut représenter l'index **i** de v_i de la manière suivante :

$$v_i = v_{k,l} \textit{ tel que } i = 6k + l \quad (0 \leq k \leq 7 \text{ et } 0 \leq l \leq 5)$$

On peut immédiatement donner une expression analytique de la permutation d'expansion **E**.

v_i (bit d'entrée) donne r_s (bit de sortie)

si $i = 6k + 1$ alors $s = (4k - 1 + 1) \bmod 32$

c) La permutation **PC1**

On peut écrire la table de PC1 de la façon suivante :

57	49	41	33	25	17	08	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36				
63	55	47	39	31	23	15	07
62	54	46	38	30	22	14	06
61	53	45	37	29	21	13	05
				28	20	12	04

Si on permute les 4 dernières lignes en mettant la dernière à la place de la cinquième, l'avant-dernière à la place de la sixième, etc ..., on obtient la table suivante :

57	49	41	33	25	17	08	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36				
				28	20	12	04
61	53	45	37	29	21	13	05
62	54	46	38	30	22	14	06
63	55	47	39	31	23	15	07

On remarque immédiatement la similitude entre cette table et celle de la permutation initiale **IP**. Les caractéristiques analytiques de la table de **IP** peuvent s'appliquer à la table de la permutation **PC1**.

5.5 Simplification de la programmation

5.5.1 Précalcul des clés

Si on dispose de suffisamment de mémoire, il est très intéressant pour améliorer les performances du **DES** et simplifier sa programmation, d'effectuer un pré-calcul des 16 sous-clés au moment de la première utilisation de l'algorithme. La clé principale restant la même durant un certain nombre de chiffrements, les sous-clés générées à chaque éxécution de l'algorithme seront identiques.
La place mémoire minimum requise est de 16 * 48 bits (96 octets) pour un gain de temps d'environ 30 % par exécution de l'algorithme DES.

5.5.2 Intégration de la permutation **P** dans les **S-Boxes**

Au lieu d'effectuer après le passage dans les **S-Boxes** la permutation **P**, on peut pour chaque **s-box** créer une table dont le contenu donnerait le résultat de la permutation des 4 bits de sortie de cette **s-box** par **P**. Les 4 bits de sortie de chaque **s-box** serviraient d'adresse pour obtenir le résultat contenu dans la table (voir figure 5.4)
Pour chaque **s-box**, le résultat obtenu consiste en un registre de 32 bits. Le résultat final sera obtenu en faisant un **OR** entre les 8 registres de 32 bits (un registre résultat par **s-box**).
Il est nécessaire pour réaliser cette table d'utiliser 512 octets mémoire (32 bits * 16 par **s-box** et 8 **S-Boxes**)

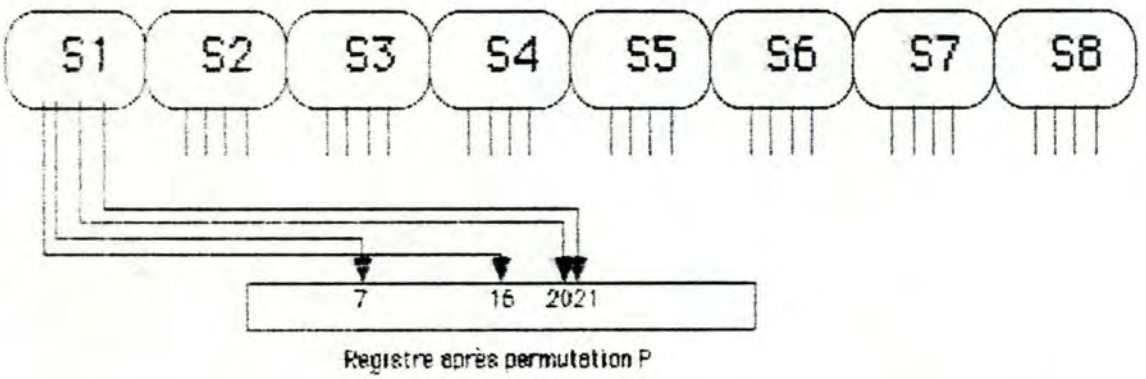


Figure 5.4 Intégration de la substitution **P** dans les **S-Boxes**

5.5.3 Combinaison substitution par **s-box** et permutation **P**

Par une démarche identique à celle ci-dessus, on peut considérer les entrées des **S-Boxes** comme des adresses dans une table contenant le résultat du passage de ces entrées à travers les **S-Boxes** et la permutation **P**.
La taille réservée pour la table devra alors être de 2K octets (32 bits * 64 par **s-box** et 8 **S-Boxes**).

5.5.4 Simplification de la permutation E

En entrée, la permutation **E** reçoit 32 bits que nous numéroterons de 1 à 32

1 2 3 4 5 6 7 8	9 10 11 .. 14 15 16	17 18 19 ..22 23 24	25 26 27 ..30 31 32

Il existe une astuce simple pour obtenir l'équivalent de la permutation **E** : au lieu d'utiliser des tables, il suffit de faire des shifts circulaires sur les 32 bits tantôt à gauche, tantôt à droite et de préserver les résultats.

On effectue d'abord un shift circulaire gauche de 1 bit sur les 32 bits en entrée de **E**, le résultat sera :

2 3 4 .. 7 8 9	10 11 12 ..15 16 17	18 19 20 ..23 24 25	26 27 28 ..31 32 1
2ème mot résultat	4ème mot résultat	6ème mot résultat	8ème mot résultat

On effectue ensuite un shift circulaire droit de 3 bits sur les 32 bits en entrée de **E**, le résultat sera :

30 31 32 1 .3 4 5	6 7 8 .. 11 12 13	14 15 16 ..19 20 21	22 23 24 ..27 28 29
1er mot résultat	3ème mot résultat	5ème mot résultat	7ème mot résultat

Les six derniers bits des 8 octets constituent l'équivalent de la permutation **E** : le mot 1 du résultat se trouve dans les six derniers bits du premier octet obtenu après le triple shift circulaire droit, le mot 2 du résultat se trouve dans les six derniers bits du premier octet obtenu après le shift circulaire gauche, ...

Chapitre 6 Alternatives à la présentation classique du RSA

6.1 Multiplication modulo n

Dans la procédure de chiffrement/déchiffrement, la routine la plus utilisée est celle de la multiplication modulo **n**.
Il est possible d'utiliser un algorithme effectuant simultanément les opérations de multiplication et de division modulo **n**.

RM =: 0
pour i = 0 jusque n-1 faire

si A_i = 1 alors RM =: RM + B
si RM > N alors RM =: RM - N
B := B + B
si B > N alors B := B - N

L'avantage de cet algorithme est de réaliser l'opération de multiplication modulo **n** en n'utilisant que des opérations d'addition et de soustraction simples à programmer en langage machine.

6.2 Réduction de la taille de la clé de chiffrement

Selon Michelman, le processus de chiffrement peut être simplifié par l'emploi d'une clé de chiffrement de 2 bits. Bien que possible, un tel système n'est cependant pas fiable car il faciliterait, pour un adversaire, la compréhension de certains blocs du message, sans que celui-ci connaisse la clé de déchiffrement.

Supposons **e = 3, n = 35**, si le premier bloc du message est **3**, le chiffrement de ce bloc donnera $C = 3^3 \pmod{35} = 27$.

Il n'y aurait pas réduction par **n** donc il sera possible de retrouver le premier bloc du message en prenant simplement la racine cubique de **27** (opération inverse de l'exponentiation par **e**)

6.3 Simplification de la procédure de chiffrement/déchiffrement

Il est possible de réduire le temps d'exécution du processus de chiffrement/déchiffrement par l'utilisation de l'algorithme suivant basé sur le théorème du reste chinois. Montrons l'intérêt de ce théorème pour le processus de déchiffrement. Introduisons d'abord quelques notations relatives aux nombres m_i, c_i, d_i.

$$c_1 = c_i \pmod p$$
$$c_2 = c_i \pmod q$$

$$d_1 = d \pmod{p-1}$$
$$d_2 = d \pmod{q-1}$$

$$m_1 = m_i \pmod p = {c_1}^{d_1} \pmod p$$
$$m_2 = m_i \pmod q = {c_2}^{d_2} \pmod q$$

Etant donné p et q, $p < q$, prenons A un entier constant tel que :

$$0 < A < q-1 \quad \text{et} \quad A = 1 \pmod q$$

Cette constante A sera obtenue par l'utilisation de l'algorithme d'Euclide donnant le plus grand commun diviseur entre p et q.
Le théorème du reste chinois nous donne immédiatement la relation suivante pour m_i :

$$m_i = (((m_2 + q - m_1)\, A) \pmod q)\, p + m_1$$

Ainsi pour déchiffrer le cryptogramme c_i, il faut d'abord calculer $m_1 = {c_1}^{d_1} \pmod p$ et $m_2 = {c_2}^{d_2} \pmod q$ plutôt que $m_i = {c_i}^{d} \pmod n$ comme c'était le cas auparavant. Les nombres c_1, c_2, d_1, d_2, ont environ 300 bits et non plus 600 comme c_i et d. Ceci permet de réduire le temps de déchiffrement d'environ **75%**. De plus, le calcul de m_1 et m_2 peut se faire en parallèle ce qui accroît le temps de recouvrement de m_i.

6.4 Génération de grands nombres premiers

6.4.1 Méthode Couvreur - Quisquater (1982 a)

On construit d'abord un ensemble S de grands nombres entiers impairs à partir d'un nombre aléatoire F, partiellement ou totalement factorisé. L'ensemble S devra contenir une grande quantité de nombres premiers.
En toute généralité, l'ensemble S peut être décrit de la manière suivante :

$S = \{ s = kF + 1 \mid k = 1, 2, \ldots, K \}$ (diverses contraintes peuvent être imposées sur F pour faciliter les tests de primalité ou de composition)

On calcule ensuite la distance maximum entre 2 nombres premiers consécutifs de cet ensemble par les formules suivantes :

La valeur moyenne de la distance maximale entre 2 nombres premiers consécutifs dans un intervalle [X, X+f] est égale à :

$$2 + (\ln X - 2)(\ln (f/\ln X) + g)$$

[g étant la constante d'Euler = 0,57721]

On peut maintenant prélever un intervalle d'une taille légèrement supérieure à la taille maximale, entre 2 nombres premiers ainsi on pourra être certain de disposer d'un sous-ensemble contenant au moins un nombre premier.
On applique une technique combinatoire (Sieve d'Erathosthème) permettant d'éliminer tous les nombres du sous-ensemble pour lesquels il y a moyen de vérifier qu'ils ne sont pas premiers.
Enfin on utilise sur les nombres restant dans le sous-ensemble une série de tests permettant de démontrer s'ils sont factorisables ou non.
Et on termine sur les nombres survivants avec des tests dits de primalité.
Les nombres restants dans le sous-ensemble peuvent être certifiés premiers.

6.4.2 Méthode probabiliste (Rivest, Shamir, Adleman (1978) et Denning (1982))

Rivest, Shamir et Adleman (1978) utilisent cette méthode pour générer des nombres premiers. Elle consiste à utiliser le symbole de Jacobi, **J(a,b)**.
Pour augmenter la sécurité de l'algorithme, on prend des précautions additionnelles dans le choix des nombres premiers **p** et **q**.

1) **p** et **q** ne doivent pas avoir le même nombre de chiffres.
2) **p-1** et **q-1** doivent être difficilement factorisables (ils doivent contenir de grands nombre premiers)
3) **pgcd (p-1,q-1)** doit être petit.

On sélectionnera **p** et **q** parmi des nombres premiers appelés nombres premiers forts, c'est-à-dire tels que :

$$p = 2p' + 1 \qquad q = 2q' + 1$$

où **p'** et **q'** sont des nombres premiers qui peuvent eux-même être des nombres premiers forts.

6.4.3 Conclusion

Des deux méthodes, la première est la seule à garantir que le nombre généré est bien un nombre premier ; la seconde assure que le nombre est premier avec une marge d'erreur très faible.
Cependant, il n'existe pas encore de version implémentable de la première méthode alors qu'il est facile de programmer la deuxième.

Chapitre 7 Application : cas Bancontact

7.1 Introduction

Chaque jour, les appareils **Bancontact** transfèrent électroniquement des millions de francs. De telles transactions ne peuvent être garanties que si les identités des usagers du système sont correctement validées et qu'aucune altération des messages n'intervient dans le réseau de communications.
Tout utilisateur du système possède une carte magnétique contenant entre autres le numéro de cette carte et la date d'expiration.
Lorsque l'utilisateur veut effectuer une opération, il introduit sa carte dans un terminal **Bancontact** (distributeur de billets, station-service ou commerce) et fournit son numéro d'identification personnel. Si ce numéro est correct, si le montant demandé n'excède pas le crédit hebdomadaire alloué et si le compte utilisateur est suffisamment alimenté, l'ordinateur central autorise le transfert de fonds. C'est-à-dire qu'il y a débit du compte du client du montant requis et crédit du compte de l'agence, station-service ou commerce de ce même montant.

7.2 Le système Bancontact

7.2.1 Introduction

Bancontact est une société coopérative qui regroupe aujourd'hui 25 organismes financiers. Elle a été créée en 1978 dans le but de développer un réseau commun de guichets automatiques bancaires (**ATM**) et de terminaux de payement dans le commerce.
Pour concevoir le système **Bancontact**, définir les solutions et les stratégies, trois objectifs ont guidé les choix :

- objectif économique : si on veut une généralisation des systèmes électroniques de transfert de fonds (**EFT**), il faut que les coûts des produits et d'exploitation soient les plus réduits possibles.

- objectif de sécurité : la technologie électronique est à la portée de personnes de plus en plus nombreuses, il fallait veiller à tous les aspects touchant à la sécurité.

- objectif lié à la stratégie des banques : le payement électronique doit rester sous le contrôle du secteur financier, c'est donc aux banquiers qu'il appartient de le développer en collaboration avec les entreprises qui veulent en bénéficier.

7.2.2 Guichets automatiques bancaires

305 guichets installés réalisent environ 1 300 000 opérations de retrait par mois. Cette moyenne très élevée indique que le client a bien accepté le principe de la carte plastique et qu'il est disposé à l'utiliser pour effectuer des payements dans d'autres secteurs d'application si cela se concrétise.

7.2.3 Secteur de la distribution de carburant

En juillet 1980, en collaboration avec le secteur pétrolier, **Bancontact** a introduit l'**EFT** dans les plus importantes stations-service de Belgique.
L'**EFT** permettait en effet de profiter des avantages suivants :

- extension et amélioration du service (24 heures sur 24, 7 jours sur 7)
- diminution des espèces et donc du risque lié à la présence de celles-ci dans les stations-service.
- diminution des coûts de distribution grâce à l'automatisme intégral.

- développement d'une image de modernisme permettant d'attirer une clientèle supplémentaire.

Près de 1000 stations-service sont aujourd'hui équipées de terminaux **Bancontact** (**PTO**) et ces terminaux réalisent mensuellement environ 1.000.000 d'opérations.
Pour des raisons de sécurité mais aussi de facilité d'intégration aux équipements existants, **Bancontact** assure la maintenance, l'installation et la fourniture des terminaux **PTO**.

7.2.4 Le secteur de la distribution

Bancontact s'est avant tout intéressé à disposer dans les hypermarchés des appareils EFT plutôt que de rechercher la clientèle des petits commerces.
Ces appareils (**pinpads**) sont directement connectés à la caisse enregistreuse. Leur contrôle, conception, développement, production et maintenance sont entièrement assurés par **Bancontact** pour les raisons expliquées ci-dessus.
Les **pinpads** utilisent des interfaces et des procédures de communication standards auxquels les constructeurs de caisses peuvent satisfaire aisément. Diverses solutions ont été mises en œuvre pour permettre l'utilisation du système **Bancontact** sur des caisses non compatibles.
Dans l'avenir, la compatibilité entre le système **Bancontact** et le système **Mister Cash** ainsi que la mise en place de **pinpads** dans les petits et moyens commerces devraient permettre une généralisation de ce mode de payement.

7.3 Equipements

7.3.1 Réseau de communications : **Banconet**

Le réseau de communications est construit à partir de lignes louées. Cela permet d'obtenir une plus grande fiabilité par rapport aux réseaux commutés (**DCS**, réseau téléphonique). Ce genre de réseau répond mieux aux besoins (mieux adapté pour une structure en étoile) et est moins coûteux que le réseau **DCS**.

7.3.2 Ordinateur central

Son premier rôle est de gérer l'ensemble des terminaux et concentrateurs ainsi que les transactions **EFT** sur base d'un fichier contenant les cartes émises. Il doit également produire les bandes magnétiques pour la chambre des compensations, assurer la liaison avec les organismes financiers et les sociétés où sont installés les terminaux de payement et effectuer des prestations complémentaires aux opérations de payement.
Le nouveau système en développement consistera à autoriser les transactions en fonction du disponible du compte attaché à la carte.

7.4 Vulnérabilité des systèmes électroniques de transfert de fonds

Tout système électronique de transfert de fonds est essentiellement constitué des quatre composants suivants : liaisons de communication, ordinateurs, terminaux et cartes magnétiques.
Pour chacun des composants, il s'agit de mettre en œuvre diverses mesures de sécurité destinées à parer toute attaque éventuelle.

7.4.1 Ordinateurs

Les applications actuelles (time-sharing, communication interactive en temps réel, communication machine à machine) permettant d'accéder à un ordinateur à partir de terminaux éloignés ou de périphériques divers. Il devient alors possible de copier, d'altérer, de remplacer ou de détruire des programmes ou des données.
Pour faire face à cela, diverses mesures sont prises telles que :
sécurité physique, protection sur les types de procédures pouvant être lancées, cryptographie, ...

7.4.2 Terminaux

La tendance actuelle est d'employer de plus en plus de terminaux et de les disposer dans un grand nombre d'endroits différents, ceci entraîne qu'il n'est plus toujours possible de garantir pour chaque terminal un bon niveau de sécurité. Ces terminaux peuvent être démontés et l'information qu'ils contiennent, peut être utilisée pour reproduire de faux messages.
Des mesures de sécurité physique (telles qu'effacement des clés si le terminal est forcé) permettent une bonne prévention des risques.

7.4.3 Cartes magnétiques

Il est relativement facile de contrefaire les cartes magnétiques car il n'est même pas nécessaire de connaître l'information qui s'y trouve pour pouvoir la copier;
La duplication est rendue difficile par le fait qu'on donne à chaque carte une propriété intrinsèque qui sera vérifiée à chaque lecture. Les nouvelles technologies ont rendu possible la création d'une carte à microprocesseur permettant de stocker directement sur la carte des informations relatives au compte client et d'effectuer les procédures d'identification sur le microprocesseur de la carte plutôt que dans le terminal. Il en résulte un

accroissement considérable de la sécurité mais les coûts d'usage de cette nouvelle carte sont encore trop élevés.

7.4.4 Liaisons de communication

Il existe de nombreux moyens de manipuler les informations sur les liaisons de communication pour commettre des fraudes.
Par exemple, on pourrait modifier le montant de la transaction, le type de transaction (remplacer un débit par un crédit), le numéro de compte débité ou encore simuler le message de l'unité centrale au terminal spécifiant que la transaction est approuvée. La plupart des fraudes consistent donc à modifier les données en temps réel. Ces fraudes peuvent être prévenues de deux manières par le chiffrement et/ou par la signature des messages.

7.5 Utilité des signatures dans le cas Bancontact

Le **DES** sera utilisé pour calculer le code d'authentification des messages échangés entre les terminaux et l'ordinateur central. Le code d'authentification était jusqu'à présent généré par une fonction secrète programmée par **Bancontact**. L'emploi du **DES** rendrait la procédure de calcul plus standard et plus sûre (la résistance aux attaques de la fonction secrète n'est pas connue comme c'est le cas pour le **DES**).
L'optimisation du temps d'exécution du **DES** est rendue indispensable pour éviter l'attente des clients aux terminaux.

Le **RSA**, encore trop lent pour générer des signatures pour **Bancontact**, serait employé pour transférer et charger dans les terminaux les clés utilisées par le **DES**.

Chapitre 8 Réalisation pratique

8.1 Introduction

Bancontact disposait déjà d'une version programmée et certifiée du DES tournant sur processeur 8-bits 6803 (voir jeu d'instructions du 6803 en annexe 4). Toutefois, cette version avait été implémentée sur base de la description DES donnée par le National Bureau of Standards et effectuait une opération de chiffrement/déchiffrement en 229 msec. Beaucoup trop lent pour être utilisé efficacement, le DES de **Bancontact** devait être reprogrammé de manière à tourner en moins de 100 msec pour une opération de chiffrement/déchiffrement.
La programmation de la version optimisée a été réalisée morceau par morceau, chaque nouvelle version de procédure était d'abord intégrée et testée au sein de l'ancien DES afin d'évaluer sa validité et son efficacité.
Par cette façon de travailler, on pouvait :

- estimer à l'avance par calcul théorique combien de cycles une procédure allait faire gagner et vérifier ensuite si le résultat obtenu était bien celui attendu.

- localiser les erreurs éventuelles dans la nouvelle procédure ou dans ses entrées-sorties, le reste du programme étant garanti correct.

A chaque étape, le version modifiée du DES était validée avec le programme DES-TEST.
Toutes les améliorations apportées devaient être prises en tenant compte des contraintes de mémoire disponible (3K octets de mémoire morte pour le programme et les tables et 256 octets de mémoire vive pour les variables et la pile) et de temps (arriver à réaliser une opération de chiffrement/déchiffrement en moins de 100 msec).
La deuxième contrainte allongeait considérablement le programme (ne pas intégrer les procédures en modules si cela allonge le temps d'éxécution) tandis que la première restreignait l'emploi d'optimisations trop coûteuses en place mémoire.

8.2 Présentation de l'algorithme DES initial

8.2.1 Programmation

Voir liste des programmes

8.2.2 Performances

Voir tableau 8.1.

Les temps donnés dans la dernière colonne du tableau représentent le temps nécessaire à l'obtention d'une opération de chiffrement ou de déchiffrement.

Wed, 4 Dec 1985, 11:52 PAGE 1

race: line#	mnemonic address	opc/data	mnemonic	break: none opcode or status	count: time, relative
after	C284	BD	JSR	cannot read data	
001	C284	BD	JSR	cannot read data	229.100 mS
002	C284	BD	JSR	cannot read data	229.244 mS
+003	C284	BD	JSR	cannot read data	229.279 mS
004	C284	BD	JSR	cannot read data	229.268 mS
005	C284	BD	JSR	cannot read data	229.267 mS
+006	C284	BD	JSR	cannot read data	229.231 mS
007	C284	BD	JSR	cannot read data	229.256 mS
008	C284	BD	JSR	cannot read data	229.279 mS
+009	C284	BD	JSR	cannot read data	229.196 mS
+010	C284	BD	JSR	cannot read data	229.244 mS
011	C284	BD	JSR	cannot read data	229.219 mS
012	C284	BD	JSR	cannot read data	229.244 mS
+013	C284	BD	JSR	cannot read data	229.364 mS
014	C284	BD	JSR	cannot read data	229.304 mS
015	C284	BD	JSR	cannot read data	229.267 mS

Tableau 8.1

8.3 Optimisations successives

8.3.1 Procédure **ROTATE**

a) Présentation

L'ancienne procédure **ROTATE** était très complexe :

- elle utilise une table de shifts de 32 octets (16 pour la table d'encryption et 16 pour la table de décryption)
- elle travaille sur des registres de 4 octets de 7 bits (le $8^{ème}$ bit n'est pas utilisé)
- elle effectue des actions différentes suivant qu'il y a déchiffrement ou non

On peut déjà obtenir une première simplification en supprimant les instructions à effectuer en cas de déchiffrement. Une seconde serait de travailler non plus avec 2 registres de 4 octets de 7 bits mais avec 2 registres de 4 octets, 3 étant de 8 bits et un de 4 bits : les shifts circulaires sur des octets de 7 bits sont plus difficiles à réaliser que sur des octets de 8 bits.
Et enfin, il est possible de transformer la table des shifts en n'utilisant plus que 2 octets au lieu de 32. En effet, l'information contenue dans la table des shifts est de type binaire, il sera facile de la condenser sur 2 octets.

b) Performances théoriques

Il fallait 752 cycles pour une exécution de la procédure **ROTATE** initiale, soit 12032 cycles pour 16 itérations.
La nouvelle procédure tourne en 221 cycles par itération soit 3544 cycles pour 16 itérations. On économise donc **8488** cycles par chiffrement/déchiffrement soit (1 cycle = 1,1 µsec) 9,3 msec.

c) Mesures

Voir tableau 8.2

Fri, 15 Nov 1985, 10:14 PAGE 1

trace: line#	mnemonic address	opc/data	mnemonic	break: none opcode or status	count: time, relative
after	C264	BD	JSR	cannot read data	
+001	C264	BD	JSR	cannot read data	219.270 mS
+002	C264	BD	JSR	cannot read data	219.270 mS
003	C264	BD	JSR	cannot read data	219.270 mS
+004	C264	BD	JSR	cannot read data	219.269 mS
+005	C264	BD	JSR	cannot read data	219.270 mS
006	C264	BD	JSR	cannot read data	219.270 mS
007	C264	BD	JSR	cannot read data	219.270 mS
+008	C264	BD	JSR	cannot read data	219.270 mS
009	C264	BD	JSR	cannot read data	219.270 mS
010	C264	BD	JSR	cannot read data	219.269 mS
+011	C264	BD	JSR	cannot read data	219.270 mS
+012	C264	BD	JSR	cannot read data	219.270 mS
013	C264	BD	JSR	cannot read data	219.270 mS
014	C264	BD	JSR	cannot read data	219.270 mS
+015	C264	BD	JSR	cannot read data	219.269 mS

Tableau 8.2

8.3.2 Pré-calcul des clés

a) Présentation

Cette optimisation est décrite au paragraphe 5.5.1. Elle a été réalisée sur l'ancienne version du DES en isolant toutes les procédures relatives au calcul des sous-clés dans une partie du programme qui sera exécutée à chaque changement de clé. L'autre partie du programme servant à réaliser le chiffrement des 64 bits de texte clair sera exécutée à chaque appel du DES.

b) Performances théoriques

Les performances sont inchangées au premier chiffrement. Par contre, aux chiffrements suivants, il ne faudra plus répéter la permutation **PC1**, 16 itérations de la permutation **PC2** et 16 itérations de la routine **ROTATE**. La durée en cycles de **PC1** est de 4523 cycles, celle de **PC2** est de 3891 cycles et celle de **ROTATE** de 221 cycles.
Le gain total en cycles par chiffrement/déchiffrement après le premier sera de (4523 + (16 * 3891) + (16 * 221)) = **73555** cycles.

c) Mesures

On vérifie que la durée du calcul des sous-clés est bien celle annoncée (**73555** cycles = **80,9** msec) et on constate une augmentation du temps d'exécution du DES proprement dit (sans calcul des sous-clés) de 9 msec due à la gestion des opérations de traitement des paramètres des diverses routines.

Voir tableau 8.3.

La première ligne du tableau donnera le temps nécessaire à la réalisation du calcul des sous-clés, les lignes suivantes indiqueront le temps d'une opération de chiffrement/déchiffrement seule (sans calcul des sous-clés).

Wed, 6 Nov 1985, 15:02 PAGE 1

race: line#	mnemonic address	opc/data	mnemonic	break: none opcode or status	count: time, relative
after	C264	BD	JSR	cannot read data	
+001	C267	BD	JSR	cannot read data	80.410 mS
+002	C267	BD	JSR	cannot read data	148.060 mS
-003	C267	BD	JSR	cannot read data	148.060 mS
-004	C267	BD	JSR	cannot read data	148.060 mS
+005	C267	BD	JSR	cannot read data	148.060 mS
-006	C267	BD	JSR	cannot read data	148.060 mS
-007	C267	BD	JSR	cannot read data	148.060 mS
+008	C267	BD	JSR	cannot read data	148.060 mS
-009	C267	BD	JSR	cannot read data	148.060 mS
-010	C267	BD	JSR	cannot read data	148.060 mS
+011	C267	BD	JSR	cannot read data	148.060 mS
+012	C267	BD	JSR	cannot read data	148.060 mS
-013	C267	BD	JSR	cannot read data	148.060 mS
-014	C267	BD	JSR	cannot read data	148.060 mS
+015	C267	BD	JSR	cannot read data	148

Tableau 8.3

8.3.3 Permutation E

a) Présentation

Cette optimisation est décrite dans le paragraphe 5.5.4. Cette nouvelle méthode de calcul de **E** est beaucoup plus simple que le recours aux tables de permutation mais requiert l'emploi de deux variables de 4 octets supplémentaires destinées à contenir les résultats partiels des shifts gauches et droits.

b) Performances

L'ensemble de la permutation **E** est réalisé maintenant en 247 cycles au lieu des 3608 cycles nécessaire à l'ancienne procédure **PERMUT**. Le gain par itération est de 3361 cycles soit de **53776** cycles (59 msec) pour les 16 itérations de **E** par chiffrement/déchiffrement

c) Mesures

Voir tableau 8.4

Fri, 8 Nov 1985, 15:40 PAGE 1

trace: mnemonic break: none count:

line#	address	opc/data	mnemonic	opcode or status	time, relative
after	C22E	BD	JSR	cannot read data	
+001	C231	BD	JSR	cannot read data	80.410 mS
+002	C231	BD	JSR	cannot read data	90.272 mS
+003	C231	BD	JSR	cannot read data	90.143 mS
+004	C231	BD	JSR	cannot read data	90.262 mS
+005	C231	BD	JSR	cannot read data	90.302 mS
+006	C231	BD	JSR	cannot read data	90.212 mS
+007	C231	BD	JSR	cannot read data	90.292 mS
+008	C231	BD	JSR	cannot read data	90.273 mS
+009	C231	BD	JSR	cannot read data	90.243 mS
+010	C231	BD	JSR	cannot read data	90.262 mS
+011	C231	BD	JSR	cannot read data	90.222 mS
+012	C231	BD	JSR	cannot read data	90.173 mS
+013	C231	BD	JSR	cannot read data	90.272 mS
+014	C231	BD	JSR	cannot read data	90.243 mS
+015	C231	BD	JSR	cannot read data	90.273 mS
+016	C231	BD	JSR	cannot read data	90.262 mS
+017	C231	BD	JSR	cannot read data	90.272 mS
+018	C231	BD	JSR	cannot read data	90.233 mS
+019	C231	BD	JSR	cannot read data	90.252 mS
+020	C231	BD	JSR	cannot read data	90.312 mS
+021	C23				

Tableau 8.4

8.3.4 Permutation **P**

a) Présentation

Cette amélioration a été décrite dans le paragraphe 5.5.2. Elle permettra de ne pas effectuer **P** par la procédure **PERMUT** mais nécessite l'usage de 512 octets de tables et 16 octets de variables pour stocker les 8 adresses des résultats.

b) Performances

Les opérations de remplacement de **PERMUT** pour la permutation **P** (analyse des sorties par **s-box**, recherche dans les tables, sauvetage des adresses successives des résultats et enfin **OR** entre les contenus des adresses sauvées pour l'obtention du résultat final) prennent 747 cycles par itération de **P** soit 11952 cycles pour 16 itérations.
Comme on n'utilise plus la procédure **PERMUT** pour effectuer **P** rapporte 38336 cycles. Le gain net est donc de **26384** cycles soit 29 msec.

c) Mesures

Voir tableau 8.5

Wed, 20 Nov 1985, 10:02 PAGE 1

race: line#	mnemonic address	opc/data	mnemonic	break: none opcode or status	count:overflow time, relative
after	C408	CC	LDD	#cannot read data	
001	C452	BD	JSR	cannot read data	79.921 mS
+002	C452	BD	JSR	cannot read data	61.763 mS
+003	C452	BD	JSR	cannot read data	61.772 mS
004	C452	BD	JSR	cannot read data	61.773 mS
005	C452	BD	JSR	cannot read data	61.782 mS
+006	C452	BD	JSR	cannot read data	61.713 mS
007	C452	BD	JSR	cannot read data	61.743 mS
008	C452	BD	JSR	cannot read data	61.782 mS
+009	C452	BD	JSR	cannot read data	61.873 mS
+010	C452	BD	JSR	cannot read data	61.773 mS
011	C452	BD	JSR	cannot read data	61.782 mS
+012	C452	BD	JSR	cannot read data	61.782 mS
+013	C452	BD	JSR	cannot read data	61.722 mS
014	C452	BD	JSR	cannot read data	61.783 mS
015	C452	BD	JSR	cannot read data	61.822 mS

Tableau 8.5

8.3.5 Permutation **PC2**

a) Présentation

Si on dispose de suffisamment de mémoire, il est très intéressant d'essayer de diminuer le nombre de cycles nécessaires à l'exécution de la permutation **PC2**. Il n'existe pas, pour cette permutation, d'expressions analytiques ou d'astuces qui permettraient d'accélérer son exécution. La seule méthode possible reste d'aller remplir tour à tour les différents mots destination avec les bits sources adéquats. Le nombre d'instructions requises est très élevé (plus de 200 au lieu de 40) mais les gains de temps sont appréciables.

b) Performances

Il ne faut plus que 471 cycles par itération de **PC2** au lieu de 3891 cycles pour une exécution de **PC2** avec l'ancienne procédure **PERMUT**.
Le gain pour 16 itérations de **PC2** est donc de **54720** cycles soit 60,1 msec.

c) Mesures

Voir tableau 8.6, ligne 1

8.3.6 Passage de l'algorithme classique à l'algorithme 48 bits

a) Présentation

Le modèle DES "48 bits" est décrit au paragraphe 5.3.2. L'implémentation de ce modèle ne donnera pas de bons résultats sur le processeur 6803 mais il peut être intéressant pour des processeurs 48-bits car il permet de ne plus utiliser certaines procédures et de diminuer le nombre d'itérations à réaliser (8 au lieu de 16)

b) Performances

Le gain observé est de 10 msec. Il est dû principalement à la simplification générale de l'algorithme, à l'abandon de la procédure **SWAP** (1836 cycles en moins), au remplacement de routines de transfert de suites d'octets par des instructions plus simples (5760 cycles en moins). Mais un plus grand nombre d'opérations est effectué (18 itérations de la permutation **E** au lieu de 16 et 2 itérations d'une nouvelle permutation E^{-1}), la place mémoire nécessaire est plus importante (création d'une nouvelle permutation E^{-1}) ainsi que le nombre de variables (24 octets RAM suplémentaires pour le stockage des résultats temporaires).

c) Mesures

Voir tableau 8.6

Wed, 27 Nov 1985, 10:15 PAGE 1

ace: ine#	mnemonic address	opc/data	mnemonic	break: none opcode or status	count: time, relative
ifter	C3D2	CC	LDD	#cannot read data	
001	C41B	CC	LDD	#cannot read data	19.290 mS
+002	C41B	CC	LDD	#cannot read data	50.666 mS
003	C41B	CC	LDD	#cannot read data	50.626 mS
004	C41B	CC	LDD	#cannot read data	50.636 mS
+005	C41B	CC	LDD	#cannot read data	50.696 mS
006	C41B	CC	LDD	#cannot read data	50.746 mS
007	C41B	CC	LDD	#cannot read data	50.746 mS
+008	C41B	CC	LDD	#cannot read data	50.676 mS
+009	C41B	CC	LDD	#cannot read data	50.675 mS
010	C41B	CC	LDD	#cannot read data	50.706 mS
011	C41B	CC	LDD	#cannot read data	50.685 mS
+012	C41B	CC	LDD	#cannot read data	50.616 mS
013	C41B	CC	LDD	#cannot read data	50.696 mS
014	C41B	CC	LDD	#cannot read data	50.796 mS
+015	C41B	CC	LDD	#cannot read data	50.696 mS

Tableau 8.6

8.3.7 Permutations **IP** IP^{-1} **PC1**

a) Présentation

Ces trois permutations, ayant des tables fort semblables, ont été révisées sur la même base. L'emploi de l'expression analytique de ces permutations donnée par Davio, ... (1983) ne pouvait pas fournir une solution valable : le gain n'aurait été que d'environ 200 cycles par permutation (600 cycles en tout) pour un nombre d'instructions trois fois plus élevé. Une autre solution consistait à prendre tour à tour les différents octets de la zone source. Avec chaque octet source, on remplissait les octets destination en tenant compte du fait que les bits d'un même octet source se retrouvent dans les bits de même indice des octets destination ou, pour le dire autrement, que les bits de même indice des octets sources se retrouvent dans le même octet destination.
Si on écrit la permutation **PC1** comme cela a été indiqué au point c) du paragraphe 5.4.2, il sera possible de la résoudre comme les permutations **IP** et IP^{-1}.

b) Performances

La nouvelle procédure réalisant la permutation **IP** ne prend plus que 655 cycles au lieu de 4840 précédemment.
La nouvelle procédure réalisant la permutation IP^{-1} ne prend plus que 629 cycles au lieu de 4840 précédemment.
La nouvelle procédure réalisant la permutation **PC1** ne prend plus que 621 cycles au lieu de 4525 précédemment.
Le gain total sera de **8396** cycles (9,2 msec) pour la partie du programme réalisant l'encryption proprement dite et de **3902** cycles (4,2 msec) pour la partie du programme effectuant le calcul des sous-clés.

c) Mesures

Voir tableau 8.7

Thu, 5 Dec 1985, 10:09 PAGE 1

trace: mnemonic break: none count:

line#	address	opc/data	mnemonic	opcode or status	time, relative
after	C6C2	FD	STD	cannot read data	
001	C307	CC	LDD	#cannot read data	13.970 mS
+002	C5BD	7E	JMP	cannot read data	40.100 mS
003	C5BD	7E	JMP	cannot read data	40.071 mS
004	C5BD	7E	JMP	cannot read data	40.100 mS
+005	C5BD	7E	JMP	cannot read data	40.071 mS
+006	C5BD	7E	JMP	cannot read data	40.100 mS
007	C5BD	7E	JMP	cannot read data	40.071 mS
+008	C5BD	7E	JMP	cannot read data	40.100 mS
+009	C5BD	7E	JMP	cannot read data	40.071 mS
010	C5BD	7E	JMP	cannot read data	40.100 mS
011	C5BD	7E	JMP	cannot read data	40.071 mS
+012	C5BD	7E	JMP	cannot read data	40.100 mS
013	C5BD	7E	JMP	cannot read data	40.071 mS
014	C5BD	7E	JMP	cannot read data	40.100 mS
+015	C5BD	7E	JMP	cannot read data	40.071 mS

Tableau 8.7

8.3.8 Retour à l'algorithme DES "32 bits" et combinaison substitution par **s-box** et permutation **P**

a) Présentation

Il s'agit de la transformation proposée au paragraphe 5.5.3. Elle nécessite l'emploi de 2 Kbytes de mémoire pour stocker les tables **S** (mais on pourra se passer des tables **P** décrites précédemment ainsi que des anciennes tables **S**).
Le texte du programme pourra également être raccourci par le fait que la procédure réalisant la substitution ne sera plus utilisée et que la nouvelle procédure combinant la substitution avec la permutation **P** sera plus simple que l'ancienne procédure réalisant la permutation **P**.
L'algorithme 48 bits est abondonné car le modèle classique 32 bits est plus simple d'utilisation sur des petits processeurs.

b) Performances

On remplace 16 itérations de **S-SUBS** (7344 cycles) et de **P_PERM** (11952 cycles) par 16 itérations de **SUB-P-PERM** (11568 cycles). On gagne encore 2 itérations de la permutation **E** (494 cycles) et de la permutation E^{-1} (604 cycles).
On simplifie l'algorithme général de chiffrement (8547 cycles) par une nouvelle version "32 bits" (6924 cycles).
Le gain global est de **11493** cycles soit 11,4 msec.

c) Mesures

Voir tableau 8.8

Tue, 29 Jan 1985, 11:41 PAGE 1

race: line#	mnemonic address	opc/data	mnemonic	break: none opcode or status	count: time, relative
after	C000	00	NOP	#cannot read data	
001	C200	CC	LDD	#cannot read data	13.823 mS
+002	C9FB	CE	LDX	#cannot read data	26.737 mS
003	C9FB	CE	LDX	#cannot read data	26.747 mS
004	C9FB	CE	LDX	#cannot read data	26.767 mS
+005	C9FB	CE	LDX	#cannot read data	26.707 mS
+006	C9FB	CE	LDX	#cannot read data	26.787 mS
007	C9FB	CE	LDX	#cannot read data	26.767 mS
.008	C9FB	CE	LDX	#cannot read data	26.667 mS
+009	C9FB	CE	LDX	#cannot read data	26.727 mS
010	C9FB	CE	LDX	#cannot read data	26.726 mS
011	C9FB	CE	LDX	#cannot read data	26.687 mS
+012	C9FB	CE	LDX	#cannot read data	26.747 mS
013	C9FB	CE	LDX	#cannot read data	26.717 mS
014	C9FB	CE	LDX	#cannot read data	26.776 mS
+015	C9FB	CE	LDX	#cannot read data	26.767 mS

Tableau 8.8

8.4 Evaluation théorique globale du programme

8.4.1 Performances

a) Procédure de pré-calcul des sous-clés

Calcul des temps en cycles

- **CALKEY** : 629 cycles
- **PC1_PERM** : 621 cycles
- **PC2_PERM** : 470 cycles (16 itérations soit 7520 cycles)
- **ROTATE** : 221 cycles (16 itérations soit 3544 cycles)

Total = **12314** cycles soit **13,6** msec

b) Procédure de chiffrement

Calcul des temps en cycles

- **DEA** : 6294 cycles
- **IP_PERM** : 655 cycles
- **E_PERM** : 247 cycles (16 itérations soit 3936 cycles)
- **SUBPPERM** : 666 cycles (16 itérations soit 10656 cycles)
- **IP^{-1}_PERM** : 620 cycles

Total = **22791** cycles soit **25** msec

8.4.2 Espace mémoire requis

Pour l'ancien programme DES, l'espace mémoire requis était de 47 octets pour les variables, 644 octets pour les tables et 371 octets pour le programme.
Pour la nouvelle version du DES, il faut 176 octets pour les variables, 2K octets pour les tables et 993 octets pour le programme. Comme on le voit, les améliorations de performances ont augmenté considérablement la place mémoire requise.

8.5 Justification des choix de solution

Comme on aura pu le constater, certains choix d'optimisation décrits dans le chapitre 5 n'ont pas été pris en compte :

- Le gain de la table **P** (5.3.1) n'a pas été retenu car cette solution oblige à créer une nouvelle permutation **E-P** pour laquelle une expression analytique ne peut être trouvée. Donc, si on gagne 38336 cycles en n'effectuant pas la permutation **P**, on est obligé d'effectuer **E-P** avec une procédure compliquée.
  La méthode employée dans la réalisation pratique du DES a été de simplifier **E** (gain de 53776 cycles) et de combiner la permutation **P** et la substitution par **S-boxes** (gain de 7728 cycles).
  Le gain apporté par l'emploi de la permutation **E-P** est donc inférieur à celui que l'on a pu obtenir par l'emploi de deux autres techniques

- La version itérative du DES est surtout une optimisation intéressante du point de vue hardware (simplification des circuits, diminution des cross-wiring, ...) ou dans le cas d'un processeur plus performant que le 6803 (disposant d'instructions de permutation de bits par exemple). Dans le cas du 6803, l'adoption de la version itérative du DES allongerait le temps d'exécution (15 itérations supplémentaires des permutations **IP** et $\mathbf{IP^{-1}}$)

- Les caractéristiques analytiques de certaines permutations permettent de se passer de tables pour ces permutations mais elles obligent à effectuer, pour chaque bit destination, un calcul donnant l'indice du bit source et ensuite une opération de transfert visant à prélever le bit source et à le placer dans le bit destination correspondant. La combinaison de ces deux opérations rend les caractéristiques analytiques lourdes et complexes à utiliser. L'intérêt qu'elles peuvent avoir, serait beaucoup plus grand, si le processeur sur lequel on travaille, permettait d'effectuer facilement des masquages d'octets.

- Le modèle "48 bits" est utile lorsqu'on travaille sur un processeur 48 bits. Si ce n'est pas le cas, l'utilisation de ce modèle allonge les opérations (2 itérations supplémentaires de **E** et de $\mathbf{E^{-1}}$).

Chapitre 9 Conclusion

Comme le lecteur aura pu le remarquer, c'est surtout le **DES** qui est privilégié dans cette étude des signatures digitales. Il y a deux raisons à cela :

- les signatures réalisées sur base de l'algorithme **DES** (généralement des signatures arbitrées) ont une exécution plus rapide que celles basées sur le **RSA** et donc peuvent être plus facilement intégrées sur un réseau de communications existant sans pénaliser les utilisateurs par une diminution des performances.
- les signatures digitales sont à l'heure actuelle le plus fréquemment utilisées dans les systèmes électroniques de transfert de fonds et servent donc à l'authentification des messages transitant sur des équipements appartenant à la même organisation. Dans ce cas, les signatures arbitrées sur base du **DES** sont les plus appropriées.

L'apport original de ce travail est de réunir un ensemble d'optimisations inédites du **DES**. Cet ensemble est présenté de manière telle que le lecteur a à sa disposition diverses transformations possibles pour adapter un **DES** à ses besoins (modèle "48 bits" mieux adapté pour les processeurs 48 bits, intégration de la permutation **P** dans les **S-boxes** au lieu des tables de combinaisons **S-boxes - P** au cas où moins de mémoire est disponible, ...)

Les optimisations concernant le **RSA** sont relativement limitées car le sujet est suffisamment vaste pour constituer à lui seul un mémoire de fin d'études.

Diverses applications nouvelles des signatures digitales ont été mises en évidence par Chaum (1985). Ces applications concernent l'utilisation des signatures comme pseudonymes digitaux en vue de rendre toute information relative aux individus impossible à suivre " à la trace". Elles apportent donc une solution aux problèmes de l'atteinte à la vie privée constitués par le regroupement possible des fichiers informatiques d'entreprises différentes (banques, assurances, organismes publics, ...)

Annexe 1 Tables de permutation du DES

Les nombres des tables de permutation indiquent à quoi correspondent les bits de sortie de la permutation. Ainsi pour la permutation initiale IP, par exemple, la première ligne de la table est la suivante :

58 50 42 34 26 18 10 02

Elle signifie que le bit 58 de l'entrée de la permutation initiale devient le premier bit de la sortie de cette permutation, le bit 50 d'entrée devient le bit 2 de sortie, le bit 42 en entrée devient le 3ème en sortie, ... Il en va de même pour les tables des autres permutations.

Annexe 1.1 Table de la permutation initiale IP

58 50 42 34 26 18 10 02
60 52 44 36 28 20 12 04
62 54 46 38 30 22 14 06
64 56 48 40 32 24 16 08
57 49 41 33 25 17 09 01
59 51 43 35 27 19 11 03
61 53 45 37 29 21 13 05
63 55 47 39 31 23 15 07

Annexe 1.2 Table de la permutation finale IP^{-1}

40 08 48 16 56 24 64 32
39 07 47 15 55 23 63 31
38 06 46 14 54 22 62 30
37 05 45 13 53 21 61 29
36 04 44 12 52 20 60 28
35 03 43 11 51 19 59 27
34 02 42 10 50 18 58 26
33 01 41 09 49 17 57 25

Annexe 1.3 Table de la permutation "permuted choice 1"

57	49	41	33	25	17	09
01	58	50	42	34	26	18
10	02	59	51	43	35	27
19	11	03	60	52	44	36
63	55	47	39	31	23	15
07	62	54	46	38	30	22
14	06	61	53	45	37	29
21	13	05	28	20	12	04

Annexe 1.4 Table de la permutation "permuted choice 2"

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Annexe 1.5 Table de la permutation d'expansion E

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

Annexe 1.6 Table de la permutation P

16	07	20	21
29	12	28	17
01	15	23	26
05	18	31	10
02	08	24	14
32	27	03	09
19	13	30	06
22	11	04	25

Annexe 2 Tables de substitution S-BOXES

Les fonctions de substitution S reçoivent en entrée un mot de 6 bits, le premier et le dernier bit de ce mot serviront de référence vers la ligne, les 4 bits centraux serviront de référence vers la colonne. L'intersection de cette ligne et de cette colonne donnera le résultat de 4 bits. Ainsi S_1, par exemple, se présentera de la manière suivante :

n°	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Si on fournit en entrée à cette table, le nombre binaire 010111. Le premier et le dernier bit référenceront la ligne (ici la ligne sera la deuxième : ligne 01). Les 4 bits centraux référenceront la colonne (ici la colonne sera la douzième : colonne 1011). Le résultat sera le nombre décimal 11, ce qui donne en binaire 1011 (ce nombre se trouve à l'intersection de la ligne 1 et de la colonne 11)

Annexe 2.1 Table de substitution de S_1

14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Annexe 2.2 Table de substitution de S_2

15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

Annexe 2.3 Table de substitution de S_3

10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

Annexe 2.4 Table de substitution de S_4

07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

Annexe 2.5 Table de substitution de S_5

02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

Annexe 2.6 Table de substitution de S_6

12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13

Annexe 2.7 Table de substitution de S_7

04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

Annexe 2.8 Table de substitution de S_8

13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

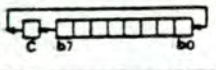
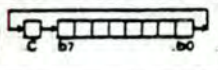
Annexe 3 Table des shifts

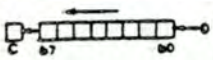
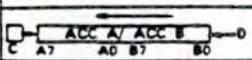
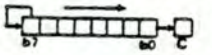
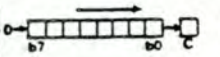
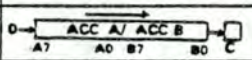
Cette table indique le nombre de shifts à effectuer, à chaque itération sur les registres C_i et D_i, lors du calcul de la clé K_{i+1}.

Numéro d'itération	Nombre de shifts gauches
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Annexe 4 Jeu d'instructions du 6803

Annexe 4.1 Instructions relatives à la mémoire et à l'accumulateur

Operations	Mnemonic	Addressing Modes															Boolean/ Arithmetic Operation	Condition Code Register					
		IMMED			DIRECT			INDEX			EXTEND			IMPLIED				5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Add	ADDA	8B	2	2	9B	3	2	AB	4	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	4	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Double	ADDD	C3	3	3	D3	4	2	E3	5	2	F3	5	3				A B + M : M + 1 → A : B	•	•	↕	↕	↕	↕
Add Accumulators	ABA													1B	1	1	A + B → A	↕	•	↕	↕	↕	↕
Add With Carry	ADCA	89	2	2	99	3	2	A9	4	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	4	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	3	2	A4	4	2	B4	4	3				A·M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	4	2	F4	4	3				B·M → B	•	•	↕	↕	R	•
Bit Test	BIT A	85	2	2	95	3	2	A5	4	2	B5	4	3				A·M	•	•	↕	↕	R	•
	BIT B	C5	2	2	D5	3	2	E5	4	2	F5	4	3				B·M	•	•	↕	↕	R	•
Clear	CLR							6F	5	2	7F	5	3				00 → M	•	•	R	S	R	R
	CLRA													4F	1	1	00 → A	•	•	R	S	R	R
	CLRB													5F	1	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	4	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	4	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Accumulators	CBA													11	1	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	6	2	73	6	3				$\overline{M} \rightarrow M$	•	•	↕	↕	R	S
	COMA													43	1	1	$\overline{A} \rightarrow A$	•	•	↕	↕	R	S
	COMB													53	1	1	$\overline{B} \rightarrow B$	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	6	2	70	6	3				00 − M → M	•	•	↕	↕	①	②
	NEGA													40	1	1	00 − A → A	•	•	↕	↕	①,	②
	NEGB													50	1	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	1	1	Converts binary add of BCD characters into BCD format	•	•	↕	↕	↕	③
Decrement	DEC							6A	6	2	7A	6	3				M − 1 → M	•	•	↕	↕	④	•
	DECA													4A	1	1	A − 1 → A	•	•	↕	↕	④	•
	DECB													5A	1	1	B − 1 → B	•	•	↕	↕	④	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	4	2	B8	4	3				A ⊕ M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	4	2	F8	4	3				B ⊕ M → B	•	•	↕	↕	R	•
Increment	INC							6C	6	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	1	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	1	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Accumulator	LDAA	86	2	2	96	3	2	A6	4	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	4	2	F6	4	3				M → B	•	•	↕	↕	R	•
Load Double Accumulator	LDD	CC	3	3	DC	4	2	EC	5	2	FC	5	3				M + 1 → B, M → A	•	•	↕	↕	R	•
Multiply Unsigned	MUL													3D	7	1	A x B → A : B	•	•	•	•	•	⑪
OR, Inclusive	ORAA	8A	2	2	9A	3	2	AA	4	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	4	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	A → Msp, SP − 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → Msp, SP − 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	3	1	SP + 1 → SP, Msp → A	•	•	•	•	•	•
	PULB													33	3	1	SP + 1 → SP, Msp → B	•	•	•	•	•	•
Rotate Left	ROL							69	6	2	79	6	3				M / A / B 	•	•	↕	↕	⑥	↕
	ROLA													49	1	1		•	•	↕	↕	⑥	↕
	ROLB													59	1	1		•	•	↕	↕	⑥	↕
Rotate Right	ROR							66	6	2	76	6	3				M / A / B 	•	•	↕	↕	⑥	↕
	RORA													46	1	1		•	•	↕	↕	⑥	↕
	RORB													56	1	1		•	•	↕	↕	⑥	↕

Operations	Mnemonic	Addressing Modes														Boolean/ Arithmetic Operation	Condition Code Register					
		IMMED			DIRECT			INDEX			EXTEND			IMPLIED				5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Shift Left Arithmetic	ASL							68	6	2	78	6	3				M / A / B 	•	•	↕	↕	⑤	↕
	ASLA													48	1	1		•	•	↕	↕	⑤	↕
	ASLB													58	1	1		•	•	↕	↕	⑤	↕
Double Shift Left, Arithmetic	ASLD													05	1	1		•	•	↕	↕	⑤	↕
Shift Right Arithmetic	ASR							67	6	2	77	6	3				M / A / B 	•	•	↕	↕	⑤	↕
	ASRA													47	1	1		•	•	↕	↕	⑤	↕
	ASRB													57	1	1		•	•	↕	↕	⑤	↕
Shift Right Logical	LSR							64	6	2	74	6	3				M / A / B 	•	•	R	↕	⑤	↕
	LSRA													44	1	1		•	•	R	↕	⑤	↕
	LSRB													54	1	1		•	•	R	↕	⑤	↕
Double Shift Right Logical	LSRD													04	1	1		•	•	R	↕	⑤	↕
Store Accumulator	STAA				97	3	2	A7	4	2	B7	4	3				A → M	•	•	↕	↕	R	•
	STAB				D7	3	2	E7	4	2	F7	4	3				B → M	•	•	↕	↕	R	•
Store Double Accumulator	STD				DD	4	2	ED	5	2	FD	5	3				A → M B → M + 1	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	4	2	B0	4	3				A − M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	4	2	F0	4	3				B − M → B	•	•	↕	↕	↕	↕
Double Subtract	SUBD	83	3	3	93	4	2	A3	5	2	B3	5	3				A : B − M : M + 1 → A : B	•	•	↕	↕	↕	↕
Subtract Accumulators	SBA													10	1	1	A − B → A	•	•	↕	↕	↕	↕
Subtract With Carry	SBCA	82	2	2	92	3	2	A2	4	2	B2	4	3				A − M − C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	4	2	F2	4	3				B − M − C → B	•	•	↕	↕	↕	↕
Transfer Accumulators	TAB													16	1	1	A → B	•	•	↕	↕	R	•
	TBA													17	1	1	B → A	•	•	↕	↕	R	•
Test Zero or Minus	TST							6D	4	2	7D	4	3				M − 00	•	•	↕	↕	R	R
	TSTA													4D	1	1	A − 00	•	•	↕	↕	R	R
	TSTB													5D	1	1	B − 00	•	•	↕	↕	R	R
And Immediate	AIM				71	6	3	61	7	3							M·IMM→M	•	•	↕	↕	R	•
OR Immediate	OIM				72	6	3	62	7	3							M+IMM→M	•	•	↕	↕	R	•
EOR Immediate	EIM				75	6	3	65	7	3							M⊕IMM→M	•	•	↕	↕	R	•
Test Immediate	TIM				7B	4	3	6B	5	3							M·IMM	•	•	↕	↕	R	•

Annexe 4.2 Instructions relatives aux registres d'index et au stack

Pointer Operations	Mnemonic	Addressing Modes													Boolean/ Arithmetic Operation	Condition Code Register					
		IMMED.			DIRECT			INDEX			EXTEND			IMPLIED				5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	5	2	BC	5	3				X−M:M+1	•	•	↕	↕	↕	↕
Decrement Index Reg	DEX													09	1	1	X − 1 → X	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	1	1	SP − 1 → SP	•	•	•	•	•	•
Increment Index Reg	INX													08	1	1	X + 1 → X	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	1	1	SP + 1 → SP	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	5	2	FE	5	3				M → X_H, (M + 1) → X_L	•	•	⑦	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	5	2	BE	5	3				M → SP_H, (M+1) → SP_L	•	•	⑦	↕	R	•
Store Index Reg	STX				DF	4	2	EF	5	2	FF	5	3				X_H → M, X_L → (M + 1)	•	•	⑦	↕	R	•
Store Stack Pntr	STS				9F	4	2	AF	5	2	BF	5	3				SP_H → M, SP_L → (M + 1)	•	•	⑦	↕	R	•
Index Reg → Stack Pntr	TXS													35	1	1	X − 1 → SP	•	•	•	•	•	•
Stack Pntr → Index Reg	TSX													30	1	1	SP + 1 → X	•	•	•	•	•	•
Add	ABX													3A	1	1	B + X → X	•	•	•	•	•	•
Push Data	PSHX													3C	5	1	X_L → M_{SP}, SP − 1 → SP X_H → M_{SP}, SP − 1 → SP	•	•	•	•	•	•
Pull Data	PULX													38	4	1	SP + 1 → SP, M_{SP} → X_H SP + 1 → SP, M_{SP} → X_L	•	•	•	•	•	•
Exchange	XGDX													18	2	1	ACCD↔IX	•	•	•	•	•	•

Annexe 4.3 Codes de condition

Operations	Mnemonic	Addressing Modes			Boolean Operation	Condition Code Register					
		IMPLIED				5	4	3	2	1	0
		OP	~	#		H	I	N	Z	V	C
Clear Carry	CLC	0C	1	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	1	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	1	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	1	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	1	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	1	1	1 → V	•	•	•	•	S	•
Accumulator A → CCR	TAP	06	1	1	A → CCR	⑩					
CCR → Accumulator A	TPA	07	1	1	CCR → A	•	•	•	•	•	•

Annexe 4.4 Instructions de saut et de branchement

Operations	Mnemonic	Addressing Modes														Branch Test	Condition Code Register					
		RELATIVE			DIRECT			INDEX			EXTEND			IMPLIED			5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Branch Always	BRA	20	3	2													None	•	•	•	•	•	•
Branch Never	BRN	21	3	2													None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	3	2													C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	3	2													C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	3	2													Z = 1	•	•	•	•	•	•
Branch If ≥ Zero	BGE	2C	3	2													N ⊕ V = 0	•	•	•	•	•	•
Branch If > Zero	BGT	2E	3	2													Z + (N ⊕ V) = 0	•	•	•	•	•	•
Branch If Higher	BHI	22	3	2													C + Z = 0	•	•	•	•	•	•
Branch If ≤ Zero	BLE	2F	3	2													Z + (N ⊕ V) = 1	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	3	2													C + Z = 1	•	•	•	•	•	•
Branch If < Zero	BLT	2D	3	2													N ⊕ V = 1	•	•	•	•	•	•
Branch If Minus	BMI	2B	3	2													N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	3	2													Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	3	2													V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	3	2													V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	3	2													N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	6	2														•	•	•	•	•	•
Jump	JMP							6E	3	2	7E	3	3				See Special Operations	•	•	•	•	•	•
Jump To Subroutine	JSR				9D	5	2	AD	5	2	BD	6	3					•	•	•	•	•	•
No Operation	NOP													01	1	1	Advances Prog. Cntr. Only	•	•	•	•	•	•
Return From Interrupt	RTI													3B	10	1		—	—	①	—	—	—
Return From Subroutine	RTS													39	5	1	See Special Operations	•	•	•	•	•	•
Software Interrupt	SWI													3F	12	1		•	S	•	•	•	•
Wait for Interrupt*	WAI													3E	9	1		•	①	•	•	•	•
Sleep	SLP													1A	4	1		•	•	•	•	•	•

Annexe 5 Tables de validation du DES

Les tests classiques du **DES** sont décrits par le **National Bureau of Standards**, dans la brochure "Validating the correctness of hardware implementation of the NBS Data Encryption Standard".
Le test du DES consiste à vérifier qu'à une clé et un texte en clair donné correspond le texte chiffré donné dans la table.

Annexe 5.1 Table de test des permutations **IP** et **E**

KEY	PLAIN	CIPHER
0101010101010101	95F8A5E5DD31D900	8000000000000000
0101010101010101	DD7F121CA5015619	4000000000000000
0101010101010101	2E8653104F3834EA	2000000000000000
0101010101010101	4BD388FF6CD81D4F	1000000000000000
0101010101010101	20B9E767B2FB1456	0800000000000000
0101010101010101	55579380D77138EF	0400000000000000
0101010101010101	6CC5DEFAAF04512F	0200000000000000
0101010101010101	0D9F279BA5D87260	0100000000000000
0101010101010101	D9031B0271BD5A0A	0080000000000000
0101010101010101	424250B37C3DD951	0040000000000000
0101010101010101	B8061B7ECD9A21E5	0020000000000000
0101010101010101	F15D0F286B65BD28	0010000000000000
0101010101010101	ADD0CC8D6E5DEBA1	0008000000000000
0101010101010101	E6D5F82752AD63D1	0004000000000000
0101010101010101	ECBFE3BD3F591A5E	0002000000000000
0101010101010101	F356834379D165CD	0001000000000000
0101010101010101	2B9F982F20037FA9	0000800000000000
0101010101010101	889DE068A16F0BE6	0000400000000000
0101010101010101	E19E275D846A1298	0000200000000000
0101010101010101	329A8ED523D71AEC	0000100000000000
0101010101010101	E7FCE22557D23C97	0000080000000000
0101010101010101	12A9F5817FF2D65D	0000040000000000
0101010101010101	A484C3AD38DC9C19	0000020000000000
0101010101010101	FBE00A8A1EF8AD72	0000010000000000
0101010101010101	750D079407521363	0000008000000000
0101010101010101	64FEED9C724C2FAF	0000004000000000
0101010101010101	F02B263B328E2B60	0000002000000000
0101010101010101	9D64555A9A10B852	0000001000000000
0101010101010101	D106FF0BED5255D7	0000000800000000
0101010101010101	E1652C6B138C64A5	0000000400000000
0101010101010101	E428581186EC8F46	0000000200000000
0101010101010101	AEB5F5EDE22D1A36	0000000100000000
0101010101010101	E943D7568AEC0C5C	0000000080000000
0101010101010101	DF98C8276F54B04B	0000000040000000
0101010101010101	B160E4680F6C696F	0000000020000000
0101010101010101	FA0752B07D9C4AB8	0000000010000000
0101010101010101	CA3A2B036DBC8502	0000000008000000
0101010101010101	5E0905517BB59BCF	0000000004000000
0101010101010101	814EEB3B91D90726	0000000002000000
0101010101010101	4D49DB1532919C9F	0000000001000000
0101010101010101	25EB5FC3F8CF0621	0000000000800000
0101010101010101	AB6A20C0620D1C6F	0000000000400000
0101010101010101	79E90DBC98F92CCA	0000000000200000
0101010101010101	866ECEDD8072BB0E	0000000000100000
0101010101010101	8B54536F2F3E64A8	0000000000080000
0101010101010101	EA51D3975595B86B	0000000000040000
0101010101010101	CAFFC6AC4542DE31	0000000000020000
0101010101010101	8DD45A2DDF90796C	0000000000010000
0101010101010101	1029D55E880EC2D0	0000000000008000
0101010101010101	5D86CB23639DBEA9	0000000000004000
0101010101010101	1D1CA853AE7C0C5F	0000000000002000
0101010101010101	CE332329248F3228	0000000000001000
0101010101010101	8405D1ABE24FB942	0000000000000800
0101010101010101	E643D78090CA4207	0000000000000400
0101010101010101	48221B9937748A23	0000000000000200
0101010101010101	DD7C0BBD61FAFD54	0000000000000100
0101010101010101	2FBC291A570DB5C4	0000000000000080
0101010101010101	E07C30D7E4E26E12	0000000000000040
0101010101010101	0953E2258E8E90A1	0000000000000020
0101010101010101	5B711BC4CEEBF2EE	0000000000000010
0101010101010101	CC083F1E6D9E85F6	0000000000000008
0101010101010101	D2FD8867D50D2DFE	0000000000000004
0101010101010101	06E7EA22CE92708F	0000000000000002
0101010101010101	166B40B44ABA4BD6	0000000000000001

Annexe 5.2 Table de test des permutations **PC1** et **PC2**

KEY	PLAIN	CIPHER
8001010101010101	0000000000000000	95A8D72813DAA94D
4001010101010101	0000000000000000	0EEC1487DD8C26D5
2001010101010101	0000000000000000	7AD16FFB79C45926
1001010101010101	0000000000000000	D3746294CA6A6CF3
0801010101010101	0000000000000000	809F5F873C1FD761
0401010101010101	0000000000000000	C02FAFFEC989D1FC
0201010101010101	0000000000000000	4615AA1D33E72F10
0180010101010101	0000000000000000	2055123350C00858
0140010101010101	0000000000000000	DF3B99D6577397C8
0120010101010101	0000000000000000	31FE17369B5288C9
0110010101010101	0000000000000000	DFDD3CC64DAE1642
0108010101010101	0000000000000000	178C83CE2B399D94
0104010101010101	0000000000000000	50F636324A9B7F80
0102010101010101	0000000000000000	A8468EE3BC18F06D
0101800101010101	0000000000000000	A2DC9E92FD3CDE92
0101400101010101	0000000000000000	CAC09F797D031287
0101200101010101	0000000000000000	90BA680B22AEB525
0101100101010101	0000000000000000	CE7A24F350E280B6
0101080101010101	0000000000000000	882BFF0AA01A0B87
0101040101010101	0000000000000000	25610288924511C2
0101020101010101	0000000000000000	C71516C29C75D170
0101018001010101	0000000000000000	5199C29A52C9F059
0101014001010101	0000000000000000	C22F0A294A71F29F
0101012001010101	0000000000000000	EE371483714C02EA
0101011001010101	0000000000000000	A81FBD448F9E522F
0101010801010101	0000000000000000	4F644C92E192DFED
0101010401010101	0000000000000000	1AFA9A66A6DF92AE
0101010201010101	0000000000000000	B3C1CC715CB879D8
0101010180010101	0000000000000000	19D032E64AB0BD8B
0101010140010101	0000000000000000	3CFAA7A7DC8720DC
0101010120010101	0000000000000000	B7265F7F447AC6F3
0101010110010101	0000000000000000	9DB73B3C0D163F54
0101010108010101	0000000000000000	8181B65BABF4A975
0101010104010101	0000000000000000	93C9B64042EAA240
0101010102010101	0000000000000000	5570530829705592
0101010101800101	0000000000000000	8638809E878787A0
0101010101400101	0000000000000000	41B9A79AF79AC208
0101010101200101	0000000000000000	7A9BE42F2009A892
0101010101100101	0000000000000000	29038D56BA6D2745
0101010101080101	0000000000000000	5495C6ABF1E5DF51
0101010101040101	0000000000000000	AE13DBD561488933
0101010101020101	0000000000000000	024D1FFA8904E389
0101010101018001	0000000000000000	D1399712F99BF02E
0101010101014001	0000000000000000	14C1D7C1CFFEC79E
0101010101012001	0000000000000000	1DE5279DAE3BED6F
0101010101011001	0000000000000000	E941A33F85501303
0101010101010801	0000000000000000	DA99DBBC9A03F379
0101010101010401	0000000000000000	B7FC92F91D8E92E9
0101010101010201	0000000000000000	AE8E5CAA3CA04E85
0101010101010180	0000000000000000	9CC62DF43B6EED74
0101010101010140	0000000000000000	D863DBB5C59A91A0
0101010101010120	0000000000000000	A1AB2190545B91D7
0101010101010110	0000000000000000	0875041E64C570F7
0101010101010108	0000000000000000	5A594528BEBEF1CC
0101010101010104	0000000000000000	FCDB3291DE21F0C0
0101010101010102	0000000000000000	869EFD7F9F265A09

Annexe 5.3 Table de test de la permutation P

KEY	PLAIN	CIPHER
1046913489980131	0000000000000000	88D55E54F54CS7B4
1007103489988020	0000000000000000	0C0CC00C83EA48FD
10071034C8980120	0000000000000000	83BC8EF3A6570183
1046103489988020	0000000000000000	DF725DCAD94EA2E9
1086911519190101	0000000000000000	E652B53B550BE8B0
1086911519580101	0000000000000000	AF527120C485CBB0
5107B01519580101	0000000000000000	0F04CE393DB926D5
1007B01519190101	0000000000000000	C9F00FFC74079067
3107915498080101	0000000000000000	7CFD82A593252B4E
3107919498080101	0000000000000000	CB49A2F9E91363E3
10079115B9080140	0000000000000000	00B588BE70D23F56
3107911598080140	0000000000000000	406A9A6AB43399AE
1007D01589980101	0000000000000000	6CB773611DCA9ADA
9107911589980101	0000000000000000	67FD21C17DBB5D70
9107D01589190101	0000000000000000	9592CB4110430787
1007D01598980120	0000000000000000	A6B7FF68A318DDD3
1007940498190101	0000000000000000	4D102196C914CA16
0107910491190401	0000000000000000	2DFA9F4573594965
0107910491190101	0000000000000000	B46604816C0E0774
0107940491190401	0000000000000000	6E7E6221A4F34E87
1907921098lA0101	0000000000000000	AA85E74643233199
1007911998190801	0000000000000000	2E5A19DB4D1962D6
1007911998lA0801	0000000000000000	23A866A809D30894
1007921098190101	0000000000000000	D812D961F017D320
10079115981901OB	0000000000000000	055605816E58608F
1004801598190101	0000000000000000	ABD88E8B1B7716F1
1004801598190102	0000000000000000	537AC95BE69DA1E1
1004801598190108	0000000000000000	AED0F6AE3C25CDD8
1002911598100104	0000000000000000	B3E35A5EE53E7B8D
1002911598190104	0000000000000000	61C79C71921A2EF8
1002911598100201	0000000000000000	E2F5728F0995013C
1002911698100101	0000000000000000	1AEAC39A61F0A464

Annexe 5.4 Table de test des **S-Boxes**

KEY	PLAIN	CIPHER
7CA110454A1A6E57	01A1D6D039776742	690F5B0D9A26939B
0131D9619DC1376E	5CD54CA83DEF57DA	7A389D10354BD271
07A1133E4A0B2686	0248D43806F67172	868EBB51CAB4599A
3849674C2602319E	51454B582DDF440A	7178876E01F19B2A
04B915BA43FEB5B6	42FD443059577FA2	AF37FB421F8C4095
0113B970FD34F2CE	059B5E0851CF143A	86A560F10EC6D85B
0170F175468FB5E6	0756D8E0774761D2	0CD3DA020021DC09
43297FAD38E373FE	762514B829BF486A	EA676B2CB7DB2B7A
07A7137045DA2A16	3BDD119049372802	DFD64A815CAF1A0F
04689104C2FD3B2F	26955F6835AF609A	5C513C9C4886C088
37D06BB516CB7546	164D5E404F275232	0A2AEEAE3FF4AB77
1F08260D1AC2465E	6B056E18759F5CCA	EF1BF03E5DFA575A
584023641ABA6176	004BD6EF09176062	88BF0DB6D70DEE56
0258161646 29B007	480D39006EE762F2	A1F9915541020B56
49793EBC79B3258F	437540C8698F3CFA	6FBF1CAFCFFD0556
4FB05E1515AB73A7	072D43A077075292	2F22E49BAB7CA1AC
49E95D6D4CA229BF	02FE55778117F12A	5A6B612CC26CCE4A
018310DC409B26D6	1D9D5C5018F728C2	5F4C038ED12B2E41
1C587F1C13924FEF	305532286D6F295A	63FAC0D034D9F793

Bibliographie

Alloin, G. (1985), *Les concepts fondamentaux d'un réseau de transfert électronique de fonds : l'approche Bancontact*, présenté à Cannes le 9 octobre.

Beker, H. J., Piper, F. C. (1982), *Cipher Systems : The protection of communications*, Northwood Books, .

Blackley, G. R., Borosh, I. (1979), "Rivest-Shamir-Adleman Public-Key Cryptosystems do not always conceal messages", *Computer and Mathematics with Applications*, volume 5, 169-178.

Chaum, D. (1985), "Security without identification : transaction systems to make Big Brother obselete", *Communications of the A.C.M.*, volume 28, n° 10, 1030-1044.

Couvreur, C. , Quisquater, J.-J. (1982 a), "Fast decipherment algorithm for RSA public-key cryptosystems", *Electronic Letters*, volume 18, n°21, 905-907.

Couvreur, C. , Quisquater, J.-J. (1982 b), "An introduction to fast generation of large prime numbers", *Philips Journal Research*, 37, n° 5-6, 231-264

Cushman, R. H. (1982), "Data-encryption chips provide security - or is it false security ?", *EDN* , Lecture notes from *International Course on Industrial Cryptography and Computer Security*.

Data encryption standard (1977), Federal Information Processing Standard Publication, n° 46, National Bureau of Standards, US Department of Commerce, Washington DC.

Davies, D. W. (1983), "Authentication and signatures of messages", Lecture notes from *International Course on Industrial Cryptography and Computer Security*, Koninklijk Universiteit van Leuven.

Davies, D. W. (1981), "Some regular properties of the Data Encryption Standard algorithm", *National Physical Laboratory*, présenté à Crypto-81

Davio, M., Quisquater, J.-J., "Principles of Cryptology", *Philips Research Laboratory Publications* , Lecture notes from *International Course on Industrial Cryptography and Computer Security.*

Davio, M., Desmedt, Y., Fosséprez, M., Goovaerts, R., Hulsbosch, J., Neutjens, P., Piret, P., Quisquater, J.-J., Vandewalle, J., Wouters, P. (1983), "Analytical Characteristics of the DES", présenté à Crypto-83.

Davio, M., Desmedt, Y., Goubert, J., Hoornaert, F., Quisquater, J.-J. (1984), "Efficient hardware and software implementation of the DES", présenté à Crypto-84.

Davis, A. J., Holdridge, D. B., Simmons, G. J., "Status report on factoring", *Sandia National Laboratory Publication.*

Denning, D. E. (1982), *Cryptography and Data Security*, Addison Wesley, Reading (Mass.).

DES Modes of Operation (1980), Federal Information Processing Standard Publication, n° 81, National Bureau of Standards, US Department of Commerce, Washington DC.

Desmedt, Y., Quisquater, J.-J. (1985), "A introduction to the security and implementation aspects of the DES", notes de cours.

Desmedt, Y. (1984), "An exhaustive key-search machine breaking one million DES keys", notes de cours.

Desmedt, Y., Vandewalle, J., Govaerts, R. (1983), "Does public-key cryptography provide a practical and secure protection of data storage and transmission", *International Carnahan Conference on Security Technology*, Zurich.

Diffie, W., Hellman, M. E. (1977), "Exhaustive Cryptanalysis of the NBS Data Encryption Standard", *Computer*, volume 10, n° 6, 74-84.

Diffie, W., Hellman, M. E. (1976), "New directions in cryptography", *IEEE Transactions on Information Theory*, IT-22, 644-654.

Financial Institution Message Authentication (1982), American Bankers Association, norme ANSI X9.9, Washington.

Gifford, D. K. (1982), "Cryptographic sealing for information secrecy and authentication", *Communications of the A.C.M.*, volume 25, n° 4, 274-286

Ingemarsson, I. (1982), "Public-key algorithm", Lecture notes from *International Course on Industrial Cryptography and Computer Security* Koninklijk Universiteit van Leuven.

Matyas, S. M. (1979), "Digital signatures - An overview", *Computer Networks*, volume 3, n° , 87-94

Meushaw, R. V. (1979), "The standard Data Encryption Algorithm", *Byte*, mars 1979, 66-76 et avril 1979, 110-126.

Meyer, C. H., Matyas, S. M. (1982), *Cryptography, a new dimension in computer data security*, J. Wiley, New-York.

Needham, R. M., Schroeder, M. D. (1978), "Using encryption for authentication in large networks of computers", *Communications of the A.C.M.*, volume 21, n° 12, 993-999.

Rivest, R. L., Shamir, A., Adleman, L. (1978), "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the A.C.M.*, volume 21, n° 2, 120-126.

Saltzer, H. J. (1978), "On digital signatures",*Operating Systems Review*, volume 12, n° 2, 12-14

Validating the correctness of hardware implementation of the NBS Data Encryption Standard (1977), National Bureau of Standards, Special Publication, Washington.